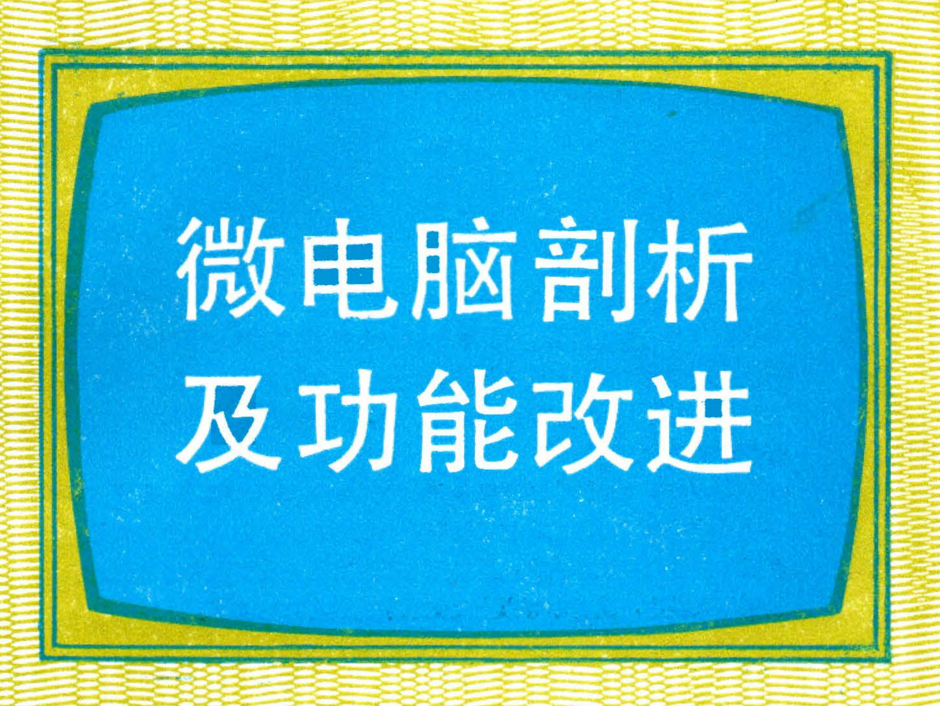
微电脑剖析及功能改进

```
10   REM  ELECTRONIC ORGAN
20   FOR A = 770 TO 795: READ B: POKE
      A,B: NEXT A
30   DATA 172,1,3,174,1,3,169,4,32
      ,168,252,173,48,192,232,209,
      253,136,208,239,206,0,3,208,
      231,96
50   DIM A(500)
60   HOME : PRINT "BEGIN TO PERFOR
      M OR PLAY,PRESS<B>OR<P>"
70   INPUT BP$
80   IF BP$ = "B" THEN 1000
90   IF BP$ = "P" THEN 2000
100   PRINT  CHR$ (7); CHR$ (7): GOTO
      70
1000  HOME : PRINT "CHARACTERS AN
      D TONES TABLE"
1010  FOR I = 1 TO 37: PRINT "=";
      : NEXT I: PRINT
1020  PRINT "!A S D O W
      1 2 3 4 5 6 7 8!
1030
105
1060  PR
1070  FOR I =
      : NEXT I: P
1080  HTAB 20: VTA       RINT "PR
```



中国青年出版社







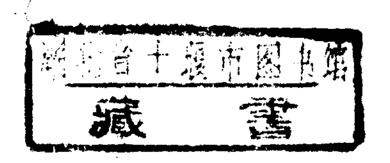
中国青年出版社

内 容 提 要

本书通过对普及型电子计算机(学习机)硬、软件系统的剖析，详细讲解了它们的构成和工作，指导读者自己动手对微电脑“开刀”进行改造，提高其性能和使用价值。对近几年在国内推广应用的功能扩充系统PH1.3的奥秘作了彻底“曝光”，供读者在进行计算机“二次开发”时参考。

书中关于BASIC系统程序文本的详细解说、PAL制视频显示系统的结构和工作原理、初级微电脑系统功能扩充方法、微电脑硬系统局部改造的实验等内容，是其他电子计算机普及读物未曾有过的，讲解深入浅出，具有中等以上文化水平的读者都能读懂。

目　　录


藏書

前　　言

做电脑真正的主人

读者朋友，你既然有兴趣翻开此书，想必是一位电脑爱好者了。你能熟练地上机操作，会编程序，或许还在计算机竞赛中获得过名次。不过，恕我冒昧问一句："你真正了解你的电脑吗？它真的听从你使唤吗？"一次，我曾这样询问一些中小学生计算机竞赛的优胜者。大多数回答是肯定的："这还用问？太了解了！""这样说吧，电脑是我最忠实的仆人。"……为了证实，我补充了一个小小的问题："你在键盘上按下'A'键的时候，为什么显示器荧光屏上就会出现一个'A'字符？这中间电脑里面究竟干了些什么？"结果，谁也没能说得清楚。于是我再问道："那么谁能从键盘敲入这样一条命令——'OFF'，让电脑关掉自己的电源？"同学们面面相觑，谁也不能。

是的，我们说了解电脑，不仅仅是熟悉它的性能和操作，还应该清楚它的内部构造和工作机制；不是只知道输入什么信息就会产生怎样的结果，而且还应该知道为什么和如何导致这一结果的。同时，如果我们只能迁就电脑那些约束使用者的规则，而不能丝毫违拗它的"脾气"，越雷池一步，那我们还远远谈不上是电脑的主人。

用户有必要这样深入地了解计算机吗？很多人认为无此必要，就像公共汽车乘客不必了解汽车原理一样。我们的观点却与此相反。如果仍然以汽车类比，那么电脑用户并不都等于"乘客"，他们很多人是"司机"或者"车主"。谁都知道，仅仅会"开汽车"的人决不能算是一个好司机。只有"懂汽车"的司机，才能使它在速度、油耗、安全、寿命各方面都达到最佳状态。计算机"司机"也是这样。

需要着重指出，计算机和汽车之类的机器有一个很大的不同之处，就是它的可开发性。计算机的使用性能并不是在制造出厂时就最后定局了的。由于系统采取的模块化和开放式结构，使之具有一定的可扩充和可重构性，给用户留有很大的用武之地。厂家提供的面向一般用户的基本使用系统，并不可能将机器的潜力发挥殆尽。在只会按使用手册"开车"的用户面前，计算机系统是个浑然一体的东西，如果某些方面不能满足需要，也只得考虑更换机型。而深知计算机结构的人，就能像庖丁那样"目无全牛"，把任何一台有模有样的电脑都看成具有可塑性的"半制成品"。他们可以采用各种手段进行"二次开发"，直接调动机器的硬软资源，增加某些功能以满足自己的需要。当电脑出现故障时，凭借他的知识便可作出大致的诊断，有的故障完全可以自己动手排除。只有这样的用户，才算得上把电脑用活了。

如果你是一位电脑迷，有志于计算机科技事业，那就更不应满足于那种"不知其所以然"的状况了。计算机科学是一门实验科学，是从一代代机器的设计制造中发展起来的，每一台电脑都物化着其中的精髓。所以，"读懂"你手中的机器是进入计算机科学殿堂的捷径。即使并不想成为电脑专家的人，研究一下计算机的原理也很有意义。令人感到神秘莫测的电脑，原来是由一些简单的元件按照并不深奥的法则构筑而成。对它的剖析可以给人很多启迪——逻辑的

方法，辩证的哲理，智力的锻炼以及科学美的感受……

事实上，很多富于求知欲和好奇心的电脑爱好者，早已不能容忍手中的电脑还是一只不知底里的“黑箱子”。有的“冒险”打开机壳看个究竟，有的煞费苦心地从头解读系统程序。但那密如蛛网的印刷电路，深不可测的集成块，“天书”般庞杂的机器语言文本，成了横亘在初学者面前的巨大障碍。他们需要引导和帮助。

给初学者讲解计算机知识，最好的办法莫过于从具体剖析一个机器系统入手。一般计算机读物，联系具体机型的多侧重于讲应用；讲解计算机原理的书又往往不结合实用机型，常以一种假设的、极度简化的“模型机”作为讨论的对象，难免使读者有隔靴搔痒之感。因此本书试图打破这种模式，以“标本机”取代“模型机”。“标本机”就是一台实际供使用的、“有血有肉”的机器。通过对它的硬软系统的深入剖析，讨论计算机是怎样构成的和怎样工作的，然后，帮助读者将知识付诸实践，对标本机进行二次开发的实验，真正体验一下作电脑“主人”的乐趣。

经过比较，我们选择了LASER 310作为标本机。

第一，它是中华学习机推广以前国内拥有量最大的学习机，电脑爱好者多数对它都很熟悉。现在很多中小学校和个人用户仍在使用。

第二，它是八位低档机型，结构和功能的复杂性适中，初学者易于理解，很适合作为解剖实例。同时，它又具有较大的可扩展性，便于动手进行实验。

第三，价格低廉(新机约三四百元，并可经调剂渠道购到旧机)，读者在经济上能够承受，进行解剖和改造的风险不大。

第四，由于它的内存小、显示分辨率低和无汉字功能等问题，正面临淘汰的边缘。作者曾为此开发过一个“LASER 310高分辨率显示及汉字系统”在国内推广。本书将以它作为扩展改造的实例详细讲解。LASER 310用户读后若能对它们进行一番改造，克服上述问题，延长应用寿命，在经济上也是有价值的。

当然，本书并不是专为LASER 310用户服务的。即使你从未接触过它，也不影响你的阅读。因为我们将对标本机的面貌、结构、功能作比较充分的描述，使之清晰地呈现在你的眼前。你可以将它同你自己所用的机器联系对比，触类旁通，达到相似的效果。

本书是电脑内部世界的一本导游手册，将引导每一位陌生的游客进入那神秘的国度，亲身游历一番，作一次实地科学考察。希望你在出口处能发出这样一声慨叹——“哦，电子计算机不过如此而已！”并且，立即着手拟订彻底征服它的计划。

祝你旅途愉快，此行成功！



第一章　解剖电脑

一、标本机概貌

本书选作剖析标本的LASER　310，是香港伟易达公司LASER系列微型电脑中的普及型学习机，也是该系列中最为成功的机型。国内不少厂家在计算机普及的第一次浪潮中引进组装，大受欢迎，一时成了学习机的主流机型之一。

LASER　310系列是逐步改进的产物，国内市场上销售过LASER　200早期型，LASER　200改进型，LASER　305和LASER　310四种型号。200型～305型的机箱较小，键盘采用类似计算器的橡胶键。310（见图1-1）改为操作方便的机械式键盘。系列发展中内部结构的改进，主要是逐步提高了逻辑器件的集成度，从而使电路得以简化；内存容量也有所增加，改善了工作条件。310是这一系列的最后一个产品，此后推出的LASER　500，LASER　3000等，已属于另外的种类了。



图1-1　LASER　310

从200型到310型，主电路板结构变化较大，但总体逻辑、系统程序和机器功能基本未变，软硬件兼容。因而，关于310型的讨论，对200型～305型也基本适用。

LASER　310机器的顶部是45键的标准英文键盘，右侧有电源开关。机箱后面是一列插

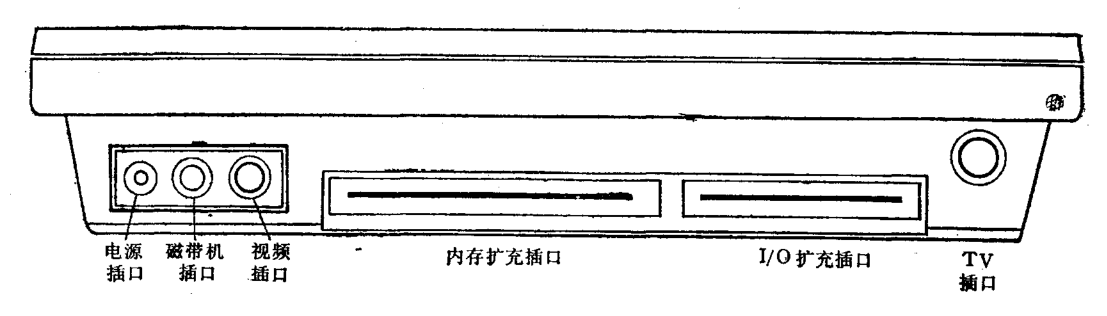


图 1-2　插口配置

口，其中有 9V 直流电源输入、磁带记录信号输入/输出、视频信号输出和射频信号输出的插孔；两个印刷电路板扩展插口中，22 线×2 的可供插接内存扩展卡或软盘驱动卡，15×2 的用来连接打印机、游戏棒或光笔等，见图1-2。

（一）　键盘功能

图1-3 是LASER　310 键盘图。它采取按键组合方式，每次按键可输入 1～5 种信息，共计可用来输入 402 种不同的字符、图符和控制信号。BASIC 语言的所有语句、命令和函数的保留词(除CLEAR)，都可一次按键输入，非常方便。这是 LASER　310 受到用户欢迎的原因之一。按键输入方式如下：

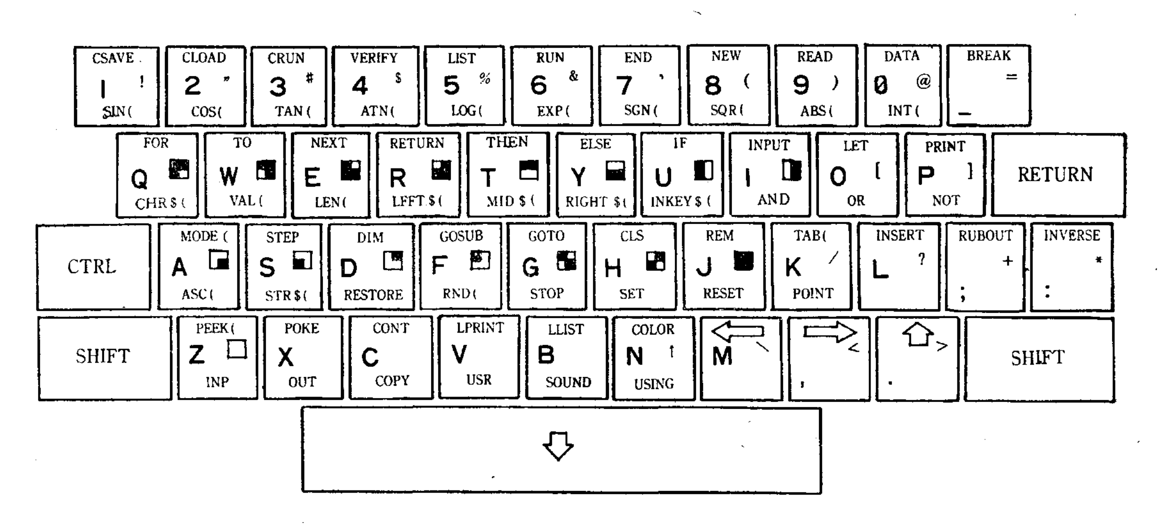


图 1-3　LASER 310 键盘

1. 每个按键均以左下部所标字符命名，单按一键即输入此字符。

2. 按住SHIFT键再按字符键(用SHIFT-口表示)，输入该键右上部所标符号或图符。图符可选择 8 种颜色之一(按键前以命令设定)。

3. 按住 CTRL 键再按字符键(用 CTRL-口表示)，输入该键上方所标保留词或控制信号。

4. 输入 FUNCTION 信号(CTRL-RETURN 键)后，单按字符键，输入每键下方所标

保留词,仅一次有效。

5. 输入INVERSE信号 (CTRL-:) 后,按上述1～4方式均输入反白字符,直至再输入INVERSE 信号或按 RETURN 键为止。

6. 每次按键输入字符时,蜂鸣器发出短促叫声则表示输入有效。

7. 按住同一键不放,即可连续输入同一字符。

LASER 310 键盘输入功能比较有特色,但没有 RESET、自定义等键,也不能输入英文小写字母。

(二) 显示功能

LASER 310有视频和射频两路输出,显示终端可选用显示器或电视机。按电视信号制式分为NTSC/PAL/SECAM三种机型。我国引进的都是PAL制机型。射频输出分为 V 频段和 U 频段两种。输出的是彩色电视信号。机箱底部有彩色/黑白选择开关。

屏幕显示有文本和图形两种模式。每次开机自动进入文本模式,以供键入程序和命令。每屏显示 16 行,每行 32 个字符。显示字符集包括全部ASCII符号和16个图案符号 (如▚▙▛等,详见"视频电路"一章)。

文本模式又分为两种显示相。正相为浅绿底黑字,负相为深绿底白字。常规开机后自动按正相显示,按住 CTRL 键开机则以负相显示。向系统工作区的 7818H 单元置入 0 或 1,也可改变屏幕显示相。在一种显示相之下,也可输入反相的字符,(用 INVERSE 键控制),但反相字符仅允许存在于字符串中。若处于引号外则不被系统接受,将报告"语法错误"。

屏幕背景颜色除上述的绿色外,还可用 COLOR 命令选择橙色背景。

图案符号用搭积木方式能用来构成低分辨率图形。每个图符(除阴影部分外)都有绿、红、蓝、黄、浅黄、青、洋红、橙等八种颜色可供选择。

图形模式用 MODE(1)语句进入,每屏 128×64 个像素,用绘图语句可以画出较细的图形。图形画面可以保持,但若转入字符显示(例如程序或命令执行结束显示"READY"表示等待输入),将自动变为文本模式,使画面清除。需要时可采用程序延时或无限循环等手段来维持画面的显示。

图形背景可取浅绿或浅黄色,每种背景上的像素颜色各有四种选择。

LASER 310 显示功能的主要缺点是分辨率低,限制了它在汉字处理和游戏方面的发展。此外,它也不支持图形/文本混合显示。

(三) BASIC功能

LASER 310 与其他学习机一样,系统软件的主体是 BASIC 解释程序,可直接由键盘输入BASIC程序和命令。每个BASIC程序行最大可容纳 64 个字符,即两个显示行的容量。每行可包含一个以上的语句(用:号分隔)。若行首有行号,系统自动将它作为程序行存贮起来;不带行号时作为立即语句(命令)执行。

合法的BASIC语句和系统命令共 40 种,函数 23 种(包括字符串函数),逻辑、算术和关系运算11种,属扩展型BASIC语言(见表7-1)。它允许使用整型(-32768～+32767)、单精度型(六位有效数字)和字符串型的变量,也支持双精度(12位有效数字)常数运算,但无双精度型变量。

当程序或命令出错时,系统会报告错误性质及出错语句所在行号,但无错误处理功能。

系统不含汇编、反汇编和管理机器语言程序的功能。用户可用POKE 语句将机器语言程序的指令代码送入内存，然后以调用 USR 函数方式启动运行。

（四） 编辑功能

LASER 310 具有全屏幕编辑功能，可用来对 BASIC 程序方便地进行修改。

它有 6 个与 CTRL 组合使用的编辑键。四个箭号键（↑ ↓ →←）可控制光标在全屏幕上下左右任意移动。光标所在的BASIC 行就成为编辑目标行。若屏上未显示此行，可用 LIST 命令列出，再将光标调入。在光标位置键入字符，即取代原字符。INSERT 是删除键，可将光标所在字符删除，其后字符依次前移补空。RUBOUT 是插入键，可在光标位置插入一个空格，其后字符依次后移。编辑键按住不放，可实现连续操作。

修改妥当后，在行中任意位置按 RETURN 键，就将全行输入程序区，取代原来同一行。这种编辑方式比中华学习机的行编辑方式简便，但不能复制和拼接。

（五） 输入/输出设备

系统基本配置的输入/输出设备是键盘、显示器（或电视机）、数据磁带机（可用普通录音机代）、压电蜂鸣器。

磁带机用来将程序或数据记存在磁带上，或将它们调入内存。其磁带记录信号是非标准形式，与别种机器不兼容。系统具有磁带读、写和校对命令，但不能控制磁带机的动作。

LASER 310有发声和音乐功能，但音频信号仅能通过压电蜂鸣片输出，音色不佳，输出的电视信号中不含音频成分。

通过机后的扩展插口，系统可以扩充多种外部设备。业已开发配套的有绘图仪（代打印机）、游戏操纵杆、软磁盘驱动器和光笔，还可加插16K/64K 内存扩展卡。用户也可利用这些插口来扩充自己所需的设备。外部设备必须带有接口电路，才能同主机联结。

二、机器的拆卸

由于微型电脑结构简单，拆卸起来并不困难。所需工具有平口和十字螺丝刀、镊子、20W 内热电烙铁等。为了防止电烙铁漏电损坏元器件，应将其外壳可靠接地。

在机器底部旋下 6 枚（有的是 8 枚）固定螺钉。然后把键盘朝上，面对后部插口，用螺丝刀轻轻撬开上下机壳。这时键盘部分同主机体分离，双方只有一条平行带状电缆相联。把键盘部分向前翻转，注意不要过分扭动电缆，以防两端焊接处脱落。

主电路板完全被金属屏蔽罩覆盖，并用 4 枚螺钉固定在下机壳上，其中一枚位于屏蔽罩的圆孔内。旋下这些螺钉，焊开连接主板屏蔽罩同底部屏蔽板的编织线，手持屏蔽罩稍向前用力使各个插孔从机壳中脱出，印板即可翻开。此时可看见印板上有一对导线同电源开关相连，另一对导线通往固定在下机壳上的压电蜂鸣片。卸下固定开关的螺钉，焊开蜂鸣片上的焊点，主板就同下机壳完全脱离。

屏蔽罩有 8 个焊脚，分别焊牢在印板的相应缺口中。将印板底面朝上，从右端开始，用烙铁顺序熔化各焊脚处的焊锡，同时用螺丝刀将焊脚撬离印板。全部游离后即可去掉屏蔽罩。电视机插口处的射频调制器小盒的顶盖也连在罩上，可一并取下。

上机壳内是键盘电路板，旋出全部螺钉，取下观察后最好立即复原，以免按键的弹簧、触点不慎脱落丢失。

LASER 310拆卸示意图见图 1-4。

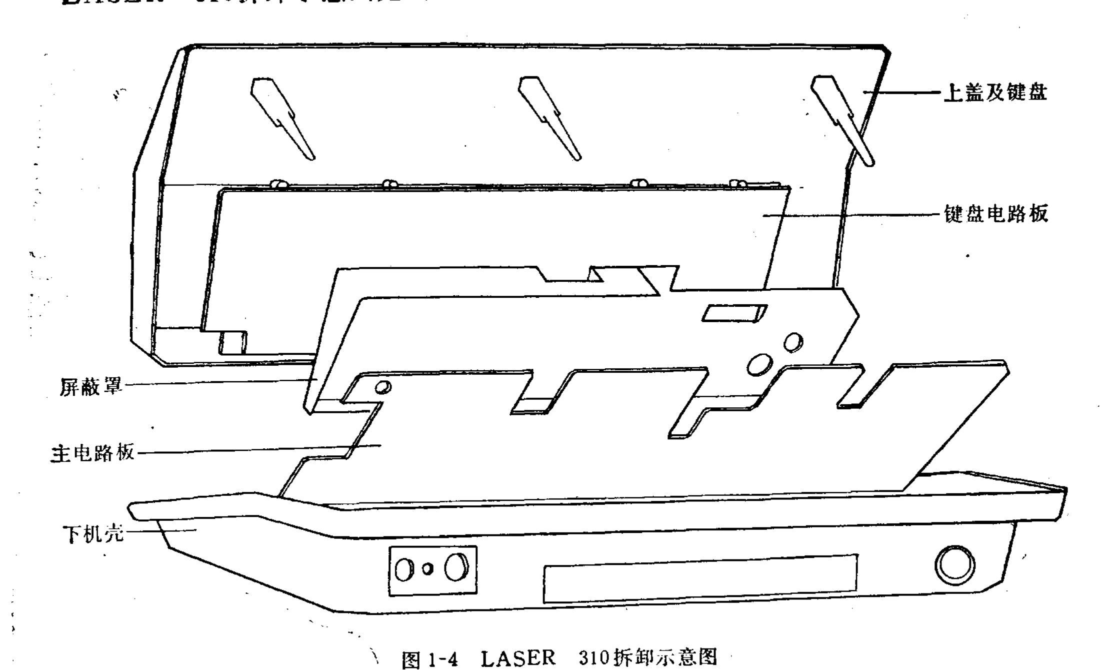
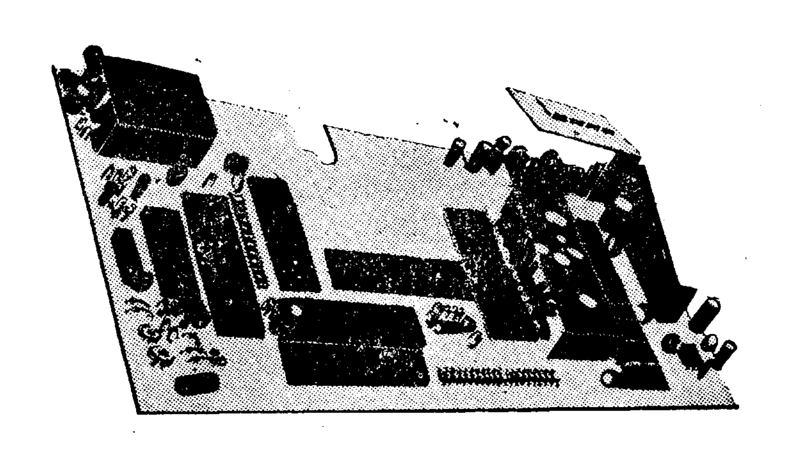


图 1-4 LASER 310拆卸示意图

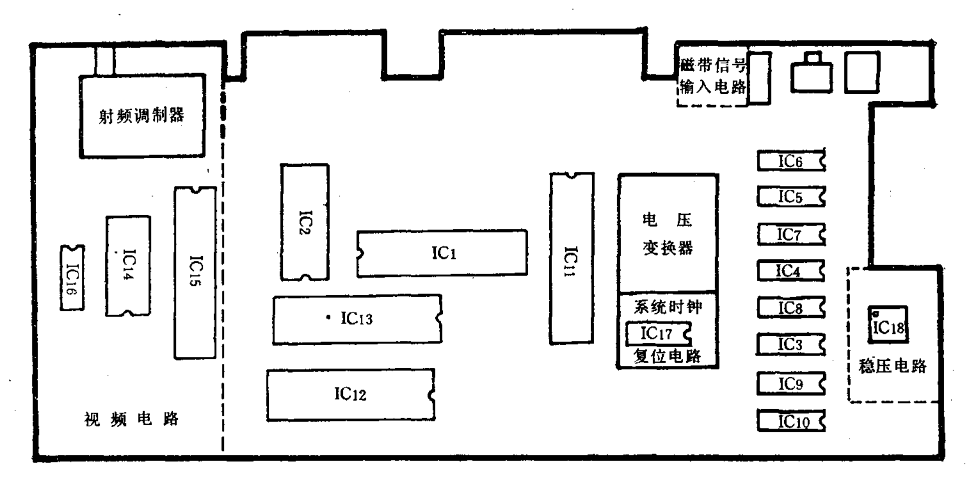
三、总体结构

机器拆开后，可以清楚地看到除键盘、蜂鸣片和开关外，所有部件都集中在一块印刷电路板上。因此，它基本上属于“单板机”类型(不过通常所说的单板机，连按键也做在同一电路板上)。

图1-5*a*是主电路板的外观。上面共有18片集成电路(IC)，还有一些由分立元件组成的功能电路单元。图 1-5*b* 是IC和功能电路单元的分布示意图。



a. 主电路板



b.主电路板布置图

图 1-5

(一) 主要集成电路

IC1 微处理器Z80A。它担任中央处理单元(CPU),除执行指令、进行运算外,还控制整个系统的工作。

IC2 只读存贮器(ROM)613128。它是系统程序存贮器,其中存放着系统软件V2.0的全部机器指令,是系统全部工作的基本依据。

IC3~IC10动态读写存贮器(DRAM)4116×8,总容量16K。作为主存贮器,用来存放用户程序、数据和系统工作信息。

IC11 门阵列电路GA008。用作DRAM接口电路,帮助CPU对主存贮器进行寻址和读写、刷新等操作。此芯片上还有一组时钟信号分频电路。

IC12 门阵列电路GA004。它是本机基本的输入/输出(I/O)设备接口电路。CPU通过它连接键盘、磁带机、蜂鸣器和视频系统,控制信息的输入或输出。

IC13 门阵列电路GA003。担任地址主译码器,对CPU发出的地址信号译码,按地址范围分别产生对ROM、VRAM和基本I/O接口的选择信号,以进行读写。此芯片中还包含有视频调整电路。

IC14 静态读写存贮器(SRAM)6116。作为显示存贮器(VRAM),容量2K。CPU把显示代码存入其中,视频系统会将它们对应的字符或图形显示于屏幕相应位置。

IC15 视频显示发生器MC6847。它周而复始地扫描VRAM,取出其中的显示代码,据以生成电视图像的亮度和色差信号,以及同步、消隐信号,供合成彩色全电视信号。

(二) 功能电路单元

1. 视频信号合成电路。

PAL制彩色编码器TBA520将6847输出的两个色差信号组合成色度信号,再由晶体管加法电路把亮度、色度和同步信号合成全电视信号输出。

2. 射频调制器(RF)。

它将上述视频信号调制到一定的电视载频(一般为二频道)上,供普通电视机接收显示。

3. 系统时钟。

由石英晶体振荡器和非门等构成,产生17.7345MHz脉冲信号,经GA003内部的分频电路分频后,变成Z80A和MC6847的时钟信号ϕ(3.457MHz),以及PAL制式的4.43MHz彩色副载频信号。

4. 电源电路。

图1-6中a部分是稳压电路单元。从外接变压整流器输入的+9V直流电,经7805集成稳压器后,稳压为+5V输出,给大部分IC供电。b部分是电压变换电路,将+9V输入分别转换成+12V和-5V电压,供DRAM使用。

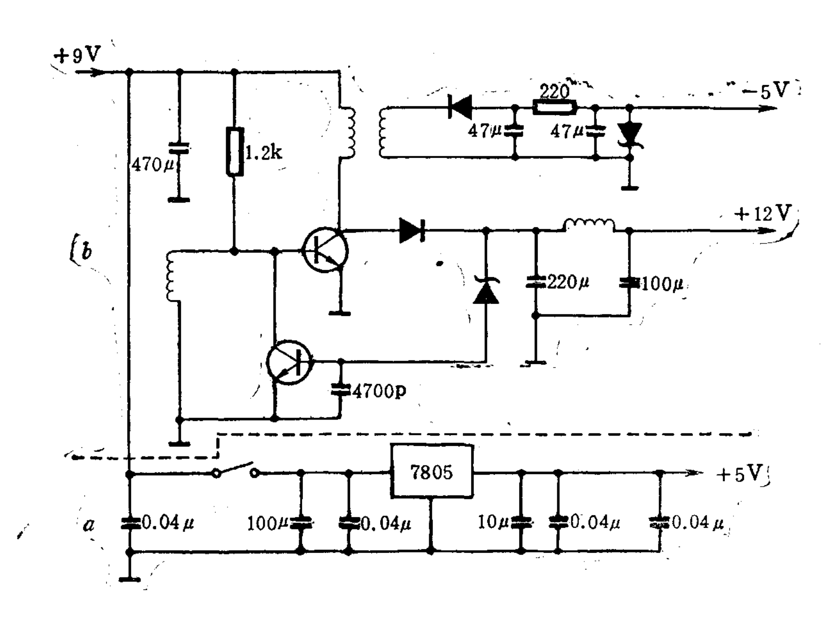


图1-6 电源电路

(三) 键盘

键盘由按键和电路板两部分重叠组成,按键的结构见图1-7。按键实质上是一些常开开关。每个键下是一对叉指状印刷电路,两端相近而不相接。当手指将键按下,键底部的导电橡胶块被压到印板上,同时接触两边的叉指端,使它们之间短路接通。手指放开,弹簧的弹力使导电橡胶块离开印板,又回到开关断开的状态。系统程序通过循环检测键盘电路,就能获知有哪个键被按下了。

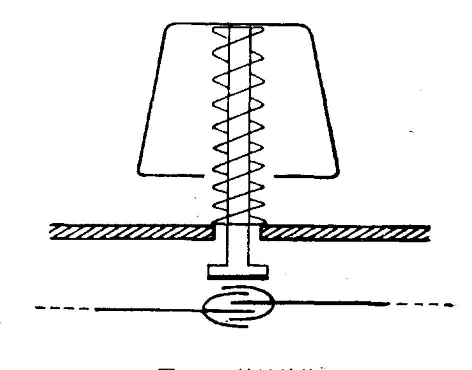


图1-7 按键结构

(四) 总体逻辑

LASER 310的总体逻辑图见第四章中图4-1(书末)。

虽然由于集成电路的使用,微型电脑的结构已大为简化,但集成电路都是多脚器件,所需连线仍然很多,线条必然细而密集,并且需交叉走线,一般采用双

面印刷电路。印板的设计和制造，难度都比较大。这也是业余电脑设计家难于独力自制一台实用型微电脑的重要原因。

CPU 需要和各个器件分别交换信息，完全采取单线直接联系的方式是不可行的，板上没有足够的布线空间。因而微型电脑的板上电路都采用“总线结构”。逻辑图上的线路比起实际的印刷电路简单得多，这是为了脉络清晰，对总线采取简化的示意法，将数据总线、地址总线和控制总线各画成一条线。从图上可以看到，同一组线可以连接多个器件，因而布线可以大为减少。

总线结构就像一个公共电话系统。同一组电话线上可以挂接多部电话机，大家都可以利用公共线路互相通话，条件是任何时候只允许两部话机占用线路。计算机器件间的“通话”也是此起彼落，可以交错进行的，而且每次“占线”时间非常短暂，所以一般不影响信息交换的及时性。

和电话系统不同的是，“通话”和“拨号”各用一组电路。地址信号相当于电话号码，译码器和接口电路相当于各级总机。CPU 先从地址总线发出地址信号，并从控制总线发出有关控制信号，各级译码器接收后产生相应的片选、行选、字选信号，就能将目标单元的数据端口打开，CPU 便可通过数据总线对它进行读写。数据总线虽然同时连接着其他器件，但此时它们都未选通，不收发数据，所以不会发生“窃听”事件。

印刷电路板边上的两组插头，就是各总线的游离端。将外部设备的相同总线和它们连接起来，就能和主机结成有机的整体，构成系统。

（五） 几点说明

1. 因组装厂和批次不同，每台 LASER 310 采用的 IC 商标、型号不尽相同，如 CPU 用日产 D780-C，VRAM 用 2016 等，但逻辑上是完全等效的。

2. 200 型～310 型电路结构的差别。

（1）200 早期型。第一，系统程序存贮器为两片 ROM；第二，未用门阵列电路，有关功能由通用 IC 组合来实现；第三，主电路板上重叠 RAM 板和 PAL 电路板各一块；第四，RAM 板上有 3 片 6116，分别用作主存和 VRAM，总容量 6K。

（2）200 改进型。第一，用 GA003 和 GA004 取代 10 片通用 IC，大大简化了电路；第二，改用单片 ROM；第三，改为单印板，板上只装有两片 6116，连 VRAM 内存仅 4K，但留有两片（2K × 2）SRAM 和一片 16KROM 的安装空位供用户扩充。

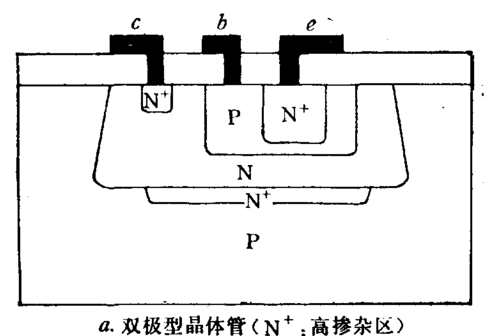
（3）305 型。内部结构与 310 型相同，将橡胶键改为机械式键盘后即成 310 型。

以下我们将对各器件进一步深入剖析。其中少数 IC，详细逻辑资料未曾发表和无从获得。对此通常可用“读者无必要了解”为理由避而不谈，但这是同本书的宗旨相悖的。我们将沿着“由表及里”的途径，尽可能引导读者深入它们的内部，并给出说明其逻辑原理的“解析图”。因为我们的目的在于“理解”而非“制造”，故不一定穷尽其所有细节，具体物理结构也未必完全相符。这一点务请读者注意。

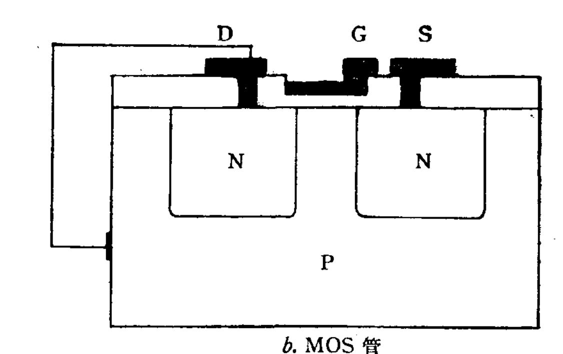
第二章 数字电路基础

标本机主印板看上去比电视机、收录机的还要简单，这是数字电路高度集成化的结果。微型计算机已经把电路的复杂性转移到集成电路内部，从而获得宏观结构的简化。为了弄清电脑的奥秘，必须“解剖”集成电路。

集成电路(IC)大都是一些矩形、多引脚的片状器件，用塑料、陶瓷或金属封装。如果有已经损坏的片子，可以打开它的外壳观察。会看到管壳内大部分面积不过是用来安置众多的引脚及连线，真正的“芯片”只占中央很小一点地方。目前使用的IC芯片都用硅晶体切片制成，边长1/4英寸(6.35mm)左右。芯片上制作了很多晶体管或阻容元件以及它们间的连线。现代计算机(包括LASER 310这样的初级学习机)普遍采用大规模集成电路(LSI)和超大规模集成电路(VLSI)，一个芯片上含有几千到几十万个晶体管，元件的表面形状已经小到肉眼不能分辨的程度。



a. 双极型晶体管(N^+:高掺杂区)



b. MOS管

图2-1 IC基本元件结构(纵剖面)

一、IC的“细胞”——开关管

(一) IC的微观结构和工艺

IC芯片的形态，既不像分立元件电路那种林立的空中结构，也不像印刷电路那样的单纯表面结构。它很像一件雕镂工艺品，是在平面上由表及里、纵深发展的立体结构。假如能用“显微外科手术”把标本机各个功能完全不同的集成电路彻底解剖开来，你将会发现组成它们的“细胞”竟是惊人的相似。各种IC的基本结构单元，可以归结为如图2-1所示的两种类型。其中*a*与普通的晶体管管芯结构相似，*b*与金属-氧化物-半导体场效应管(MOS)管芯结构相似。它们又都同样是由一些相互包围的“P区”和“N区”构成。

硅是一种半导体材料。四价的硅原子外层有四个价电子，在纯净的晶体中每个原子的价

电子都同相邻原子的价电子结成共价键，形成相当稳固的晶格。接受外界的热能、光能后，硅晶格中会有少数价电子获得足够能量而脱出共价键，成为可以移动的“自由”电子。同时，原来共价键位置成为一个可以接纳电子的“空穴”。空穴本身虽不能移动，但可通过接受电子使相邻的晶格中出现空穴，相当于空穴向电子移动的反方向移动，并可以像接力一样地依次传递。可移动的电子和空穴，都能形成电流传输电能，称为载流子。纯硅中自发产生的载流子很少，是电的不良导体。用掺入杂质的方法可以使载流子大量增加，提高导电性。硅在熔炼时如掺入微量的三价硼元素，结晶时硼原子就代替部分硅原子组成晶格。由于硼原子少了一个价电子，于是晶格中形成一个空穴。掺硼的硅晶体成为以空穴为多数载流子的P(正)型硅。如果使用的杂质是五价的磷元素，因为磷原子进入晶格同硅原子结成共价键后还会多余出一个价电子，成为自由电子，就形成以电子为多数载流子的N(负)型硅。N型硅中的少量空穴和P型硅中的少量电子是少数载流子。P型与N型的区别是由两种极性载流子的相对多数所决定的。若在P型硅中加入更多的电子(例如掺磷)，可变为N型；在N型硅中加入更多的空穴(例如掺硼)也会变为P型。集成电路就是以掺杂变型的手段，在一种硅晶体上制作出P型和N型区域互相穿插的结构。

制造IC的P型(或N型)硅晶体，直径可达10cm。将它切成0.25mm的薄片，便可以同时在上面制作几百个IC芯片。在一种衬底上制作反型区域，是通过分子扩散掺入杂质的手段实现的。为了划定掺杂区域的范围，先将基片表面全部氧化，生成二氧化硅(玻璃)保护层，然后在需要反型的部位开出“窗口”。开口的方法是用“光刻”。它类似印刷业的腐蚀制版术，在保护层上涂一层光敏抗蚀剂，上面覆盖绘有刻蚀图形的光学“底片”——掩模，凡是需要掺杂的部位都是不透明的。通过掩模给基片曝光(用紫外线或X射线)。掩模透明部分之下的抗蚀剂感光固化。用溶剂对整个表面进行腐蚀时，只有阴影部分的二氧化硅保护层被蚀刻掉，暴露出硅表面。这时把基片放到磷(或硼)蒸气中进行扩散处理，杂质原子便掺入开口部位。控制掺杂的浓度和深度，就可形成图2-1*b*那样的反型硅岛。

对于图2-1*a*中的晶体管，因为是纵向的多层次、多浓度结构，上述过程要重复多次。首先，表面氧化后用第一种掩模光刻出整个管芯面积的大孔，高浓度高深度掺磷，使整个管芯区域成为高N区；然后以较低的深度少量掺硼，使上面大部分区域再变成低N区。这个大的N区和衬底间的PN结构成与相邻管子隔离的边界。第二步，再度氧化表面，用第二掩模光刻出P区孔洞并进一步掺硼。第三步，氧化后用第三掩模光刻出两个N区并高浓度掺磷。

形成需要的PN区结构后，表面最后氧化作为绝缘层，光刻出需要接电极引线的开口。在整个表面积淀一层铝膜，使其同所有的接点实现电气接触。然后像腐蚀印刷电路板一样，光刻掉多余的铝层并实现线路分隔，把分离的元件按设计联结成有机的整体，集成电路便基本成型。基本工序见图2-2，再经切割、检测、焊线、封装，成为成品。

一个芯片上元件的多少，是由掩模设计所决定的。在一定的工艺条件下，制作元件多少生产费用相同，因此增加芯片上的元件密度可以大幅度降低每个元件的成本，还能提高工作速度和可靠性，简化整机电路结构。这些好处导致了集成规模的迅速发展。

技术上对元件密度提高的一个主要限制是光刻分辨率。当掩模线条的尺寸小到与所用光源的波长相比拟时，由于光能够衍射而“绕过”掩模图形线条，基片上不能形成所需的阴影，也就无法实现光刻了。所以，用于光刻的射线波长愈短，光刻分辨率愈高。紫外线(≤0.39微

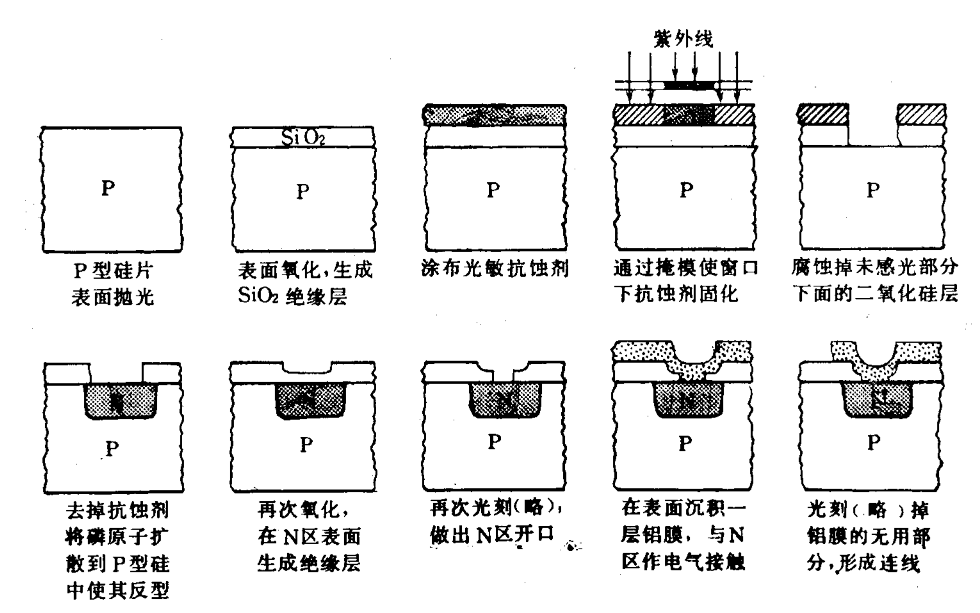


图 2-2　IC 制造工艺

米)的光刻极限≥1 微米，小于 0.04 微米的 X 射线光刻极限可达 0.1 微米。而采用电子束扫描，可以刻出更细的线条，使一块芯片上的元件数达到几十万、几百万个，不过相对成本也比较高。

(二)　开关管工作原理

数字 IC 里的晶体管和场效应管是作为开关元件来使用的。它们的工作原理需要从 PN 结的性质说起。

图 2-1 中有多处 P 型硅和 N 型硅的交界面。在这里，双方的多数载流子(N 区的电子和 P 区的空穴)互相吸引，向对侧扩散而复合。N 区一侧因少了电子而带正电荷，P 区一侧因少了空穴而带负电荷，达到动态平衡时便形成一层称为"PN 结"的静电位垒，其电场方向由 N 区指向 P 区(图 2-3)。这个电场会阻挡双方的多数载流子继续向对方扩散，对于它们相当于很大的电阻。但它同时却能推动极性相反的少数载流子通过界面，即电子可由 P 区流向 N 区和空穴可由 N 区流向 P 区。

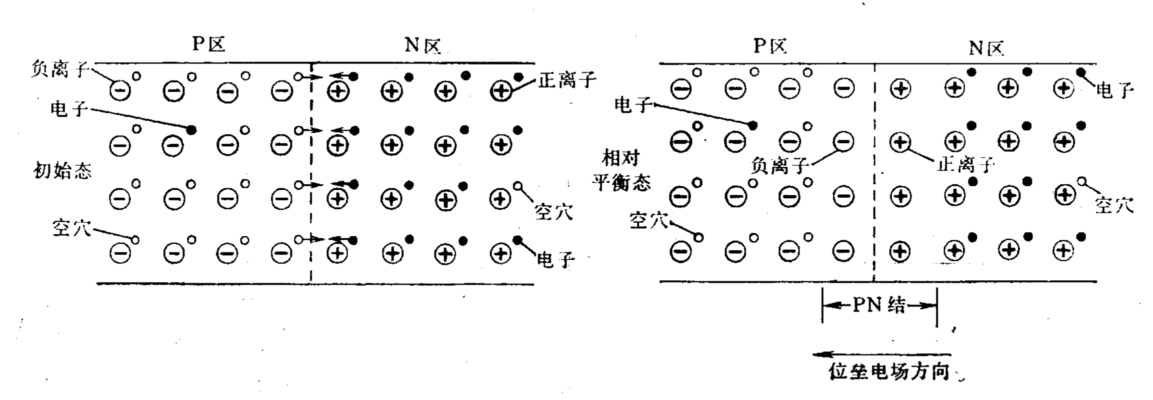
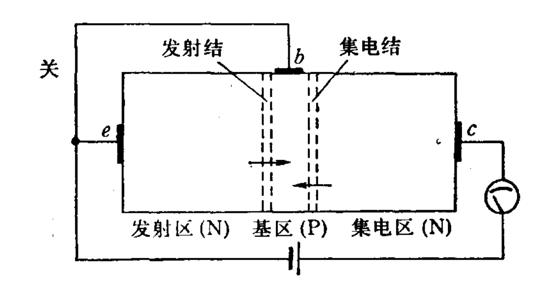


图 2-3　PN 结的形成

因为 PN 结位垒是动态平衡的，因而可外加电场加以改变。给 PN 结加上与位垒方向相反的电压(P 区为正，N 区为负)，位垒就被降低，使其对多数载流子的阻挡作用减小，相当于电阻下降，载流子在外电场的推动下较易通过，形成较大的正向电流。这样的外加电压称为正向偏压。外加与位垒方向相同的反向偏压，则使位垒更为增强，对多数载流子阻力增大，使正向电流趋于零。但同时将使少数载流子更容易通过 PN 结，形成反向电流。由于少数载流子数量甚微，一般情况下反向电流小到常可忽略不计的程度。但下面我们会看到，晶体管的工作正是利用反向电流的结果。

图 2-1 中的a,b两管都有三个电极。晶体管的发射极 e 和集电极 c，MOS 管的源极 S 和漏极 D 相当于开关的两端，而基极 b 和栅极 G 是开关的控制端(相当于按钮)。分别将 c,S 接 +5V 电源，e,D 接地(0 电位)。因为两端之间都隔着两个背靠背串联着的 PN 结，按单向导电的性质，电流似乎根本不可能同时通过相反的 PN 结从一端流到另一端。将 b 和 G 接 0 电位时测量，回路里的确无电流(少数载流子形成的很小的“漏电流”忽略未计)，管子如同断开的开关。但若将 b 和 G 改接 +5V，情况就变化了，回路中出现很大的电流，犹如开关合上一样。

晶体管和 MOS 管的开关原理是不相同的。现在先讨论较为复杂的晶体管。图 2-4 是图 2-1a 的简化。当基极 b 接近 0 电位时，b,e 间的发射结上无偏压，本身的位垒阻止 N 型发射区的多数载流子电子进入基区。c,b 间的集电结上为反向偏压，使位垒提高，N 型集电区的电子也不能通过它而进入基区。所以管子处于截止状态。即使基极电位略高于 0V，只要尚不能低消发射结位垒电场，截止状态仍可维持。



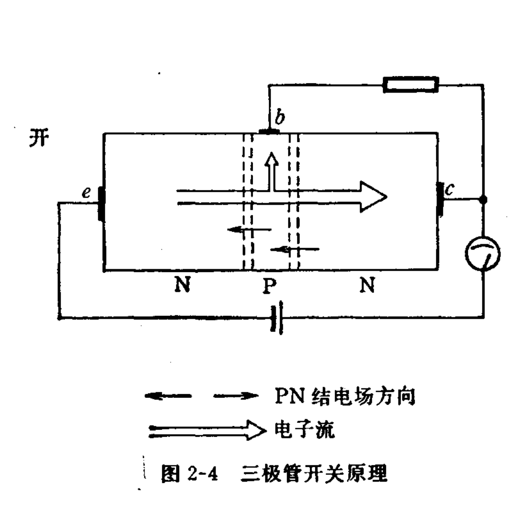


图 2-4 三极管开关原理

如果我们把基极电位提高到 1 V以上，超出发射结位垒的绝对值，抵消了它对多数载流子的阻挡作用，基区的空穴能够进入发射区，发射区的电子也能注入基区。这时，管子的几何结构将起到关键的作用。

如果把发射结看作一个二极管，越过PN结的电子流应该流入正极性的基极。但是，因为制作时把基区做得很薄。而且被N 型的集电区包围起来，集电结面积很大。由发射区进入基区的电子通过扩散作用很容易到达集电结。对于 P 型基区来说，电子是少数(尽管现在事实上已经不少)载流子，集电结位垒正适于推动它们进入 N 型集电区。集电极接电源正极，电位很高。它通过面积较大、导电性好的高浓度 N 型硅岛构成一个“电子收集器”，把进入集电区的电子吸入集电极，形成集电极电流。与此同时，进入基区的电子也有一部分扩散到距离较远、收集面较小、电位较低的基极附近，被基极吸收，形成较小的基极电流。基极偏压在一定的幅度内变化，集电极电流和基极电流保持比

例关系，可以看作是基极电流信号的放大。这时，管子工作在线性放大区。模拟电路的晶体管放大器便是工作在这种状态下。

当基极电位超过一定的高度，例如达到+2V以上，集电极电流接近最大值，不再随基极电流变化，失去线性放大作用，管子进入饱和状态。这时它的内阻很小，相当于开关的闭合状态。一般数字电路中将晶体管当作开关使用，为使开关可靠，通常在基极加以0V和≥2V的电压，使它在截止状态和饱和状态间跳变。

MOS管的开关原理比较简单。它的栅极同P型衬底间有很薄的二氧化硅绝缘层。衬底和源极S接地，漏极D接正电源。若栅极G为0电位，对衬底无影响，管子的状态同上述晶体管的截止状态相似。若使栅极变为正电位(例如+5V)，则由于静电感应作用，会吸引一批电子集中于它正下方的P型衬底表层，使局部暂时反型，形成薄层N型区，称为“沟道”。沟道使得同是N型的S区和D区连通，消除了PN结阻挡层。在外加电场的推动下，电子便可由源极到漏极顺利流动，于是管子导通。若栅极电位再次降为0V，由于电子自由扩散，N型沟道随即消失，管子再度截止。

晶体管有两种极性载流子同时参加工作，属于双极型。MOS管只有一种载流子参加工作，属于单极型。以上所举的例子说的是NPN型晶体管和增强型N沟道MOS管。这两种IC以(或主要以)电子为载流子。自由电子的运动速度比空穴的“接力”运动快，所以工作速度较高，是计算机中使用最多的类型。此外，双极型IC还有PNP型结构，它以空穴为主要载流子。单极型IC中也有P沟道增强型、N沟道耗尽型和P沟道耗尽型。耗尽型的沟道是制造时掺杂形成的，管子通常处于导通状态。栅极加上正或负电压后，驱散了同极性载流子，使沟道反型而暂时消失，变为截止状态。这些类型只是电极性不同，开关作用是一样的。

双极型IC和MOS型IC各有优缺点。双极型IC单元是纵向结构，基区厚度由掺杂技术决定，可以做得极薄，电子扩散到集电结的时间很短，故工作速度快，适用于高速器件。而MOS型IC单元是横向结构，它的沟道宽度由光刻技术决定，一般大于上述掺杂层厚度，电子穿越沟道所需时间较长，加以栅极与衬底间的电容较大，充放电也需要一定时间，所以工作速度较低。但另一方面，从结构图示可明显看出，MOS型IC单元比双极型占面积小，同样大小的硅片上可以做出四倍于双极型IC的单元电路。大容量的存贮器、复杂的微处理器等，显然以采用MOS型结构较为有利。随着集成技术的进步，各种新型IC工艺结构不断问世。

上述IC单元结构不仅可用作开关管。集成电路中的电阻器也可由晶体管或MOS管来担任，通过对PN结的偏置或沟道长宽比的设计，取得所需的(不是很精确的)电阻值。电容器则像MOS管的栅极那样制造；因面积限制，电容量不能做得很大。至于电感，目前的集成技术还无法制作在硅片上。

(三) 开关——数字计算机的基础

开关，在现代生活中是太常见、太平凡了，它的功能有限得很，一开一关而已。而电子计算机已经成为很多现代神话的主人公，几乎被认为无所不能。当听到“计算机不过是一大堆开关的有序集合”时，并不是每个人都会相信的，然而，事实就是如此。

在计算机的历史上，1944年制成的电动-机械计算机Mark Ⅰ，是第一台现代数字计算机。它的基本元件是大约3000个普通的电话继电器——可控的电流开关，工作时由于电磁铁不断吸合和释放，会像织机那样发出一片“嗒嗒”声。两年后问世的ENIAC，用真空电子管的

导通和截止来代替继电器的开关动作，从而把开关速度提高了一万倍，成为电子计算机的鼻祖。后来，晶体管开关电路进一步取代了真空管，使计算机体积、功耗都大大减小。七十年代出现的微型电脑，由集成电路组成，体积仅为ENIAC的几十万分之一，而性能却要强得多。计算机的面貌在数十年间发生了巨大的变化。然而万变不离其宗，神通广大的微型电脑中的那些集成电路，仍然由最简单的元件——开关管组成。

电子计算机的工作，主要是控制、数值运算和逻辑判断等。它们同开关有什么关系呢？

在所有控制功能中，开关控制是最基本的一种。开关管接在电路中，可以控制电路的通或断，部件的开或关，启动工作或停止，等等。

数字计算机是以数字为处理对象的。人们日常使用的十进制数有十个数码，在电子仪器里用十种不同的元件和物理状态来表示它们是困难的。因此，几乎在电子计算机诞生的同时，它的奠基人之一的冯·诺依曼就将二进制引入其中。二进制只有0和1这两个数码，实行“逢二进一”的规则，用一种双态元件便可以表示。开关，正是理想的双态元件。开关管的导通和截止，使电路的电流和电压在两种状态间跳变，形成电子形态的0和1。除了数字本身外，原则上任何信息也都可以数字化，任何数字又都可以化为二进制形式，而任何二进制数码串都可以用若干开关电路表达和处理。

电脑系统是建立在逻辑代数的基础上的。逻辑代数由英国数学家布尔于1854年创立，也称布尔代数。它是逻辑思维的数学模型，可以将逻辑问题抽象为符号演算。逻辑代数与普通代数不同的是，变量不能任意取值，而只有“1”和“0”两个——这里它们不是作为数值，而是表示命题的是或否、真或假的“逻辑量”。逻辑代数这种双值系统，正适于建立在开关的基础上。所有的逻辑电路，都由开关管构成。

由此可见，开关元件可以作为电子数字计算机的基础。电子模拟计算机，或数字计算机的声音、电视图像等模拟信号输出输入端口，才使用非开关工作方式的线性电路。

二、数字信号和数字电路

（一） 二进制

二进制同十进制的区别仅在于“基数”不同，以2代替了10。它同十进制数一样，以按位次排列的数码表示一个数，各个位上的数码表示数值是数码乘以基数的位次幂。二进制数化为十进制数可用按权展开方法，例如（后缀“B”表示二进制数）：

$$1101.101\text{B} = 1\times2^3+1\times2^2+0\times2^1+1\times2^0+1\times2^{-1}+0\times2^{-2}+1\times2^{-3}=13.625$$

将十进制数转化为二进制数，可连续除以2，将每次的余数逆行排列（最后的余数是最高位）即可。如2＝10，3＝11，4＝100……等等。

显然，二进制数有数码简单而位数较多的特点。数码简单，使得电路表达容易，基本运算规则简化。例如，二进制的“乘法表”只剩下“九九表”的第一句口诀“一一得一”（其余都得零）。这就使得机器运算很容易实现。位数多，需要的电路器件就多，运算所需的步数也必然多。而集成电路的大容量、高密度和高速度，正好适应这些要求。

但是，对于人书写、阅读来说，二进制数几乎无优点可言。它冗长单调，难写难记。因此，直接同计算机系统对话时，常使用被称为“二进制的代码”的十六进制记数法。十六进制共有十六个数码，0～9与十进制相同，用英文字母A～F作为10～15这6个数码。这种数字初

用时常感到别扭，但由于16是2的整数次幂，和二进制有很规则的对应关系，每个数码正好等于四位二进制数的值。以后缀“H”表示十六进制数，0H＝0000B，1H＝0001B，2H＝0010B，……，EH＝1110B，FH＝1111B，互相转化非常方便。例如，十六进制数8A9FH，依次将各位对应的二进制数码联在一起，就是1000 1010 1001 1111B(空格是为了说明，实际并不需要)。对于二进制数110 0001 1101B，将数码从右到左每四位划成一段，分别换成十六进制数码即成为6 1 DH。

（二） 数字信号及电路

使用任何一种计算手段，数都必须赋形于一种物质形态：笔算时是纸上的符号，珠算时是算盘珠，在计算尺上是刻度，机械式计算机里是轮齿或角度……。那么，电子计算机里的数用什么东西来表示呢？显然只能用电信号。

电的物理量如电压、电流强度等，它们的数值称为“模拟量”。直接对模拟量(例如任意的电压信号)进行运算的是模拟计算机。它运算速度快，仿真性能好，适用于对客观过程的模拟、跟踪和控制。但由于模拟信号的特点，受电路性能的限制，计算精度较低，模拟量存贮困难，信号抗干扰能力弱，适用范围比较狭窄。

绝大多数通用电子计算机都是数字式计算机。它们以数字信号为处理对象。数字信号虽同样是电信号，但其本身实际的物理量已被忽略，只是以电流的有或无、电压的高或低等两种对立的状态作为一种符号，来表示抽象的数字。例如，若以≥2V的电压表示1，以≤0.8V的电压表示0。那么，不但2V这个物理量已无本来的意义，2.23V，4.576V……也都成为相等(＝1)的了。这就像不同频率和音色的钟打的“8点”都表示同一时间，不同粗细和颜色的笔写的“3”字完全相等一样——因为它们表示同一个数字信号。

使用数字信号给计算机带来很多好处。第一，数能用来概括一切事物，因而数字式计算机具有通用性；第二，只要有足够多的位数，数值就可以达到很高的精度；第三，数字信号不易受干扰；第四，数字信号便于存贮。

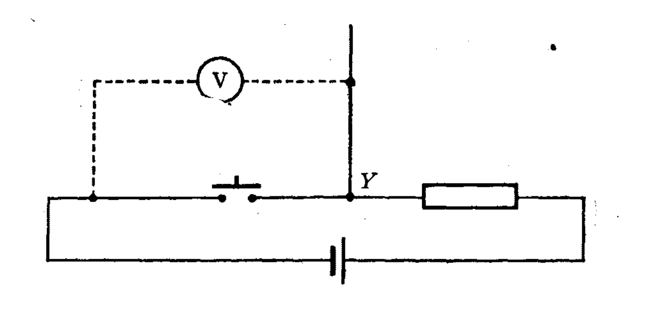
在数字计算机里，通常用电平的高和低分别表示数码1和0。电平就是电路某一点对于公共参考点(一般也就是接地点)的电位差。例如一个数据输入端的电平是0V，就是输入数据0；若是＋5V，则输入为1。采用电平信号而不是电流信号，是从电气性能考虑的。当负载电阻足够大时，很小的电流便可以产生所需幅度的电压降。若用电流信号，为了显著区分1和0，需要相当大的电流强度差，功率消耗会大得多。

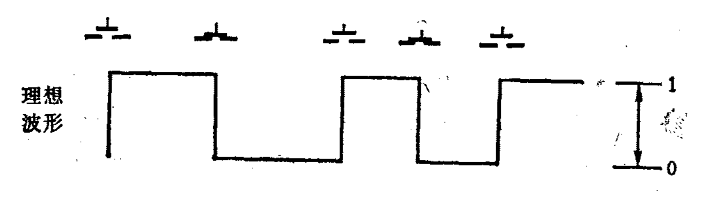
这样的数字信号很容易用开关电路来产生。图2-5的一个按键开关和一个电阻，便组成最简单的数字信号发生器。按键按下，开关闭合，Y点等于接地，电平相当于地电位(电源负极)，$Y=0$。按键开关断开，电路中无电流，Y点通过电阻接电源正极，为高电平，$Y=1$。这个电路可以用作数字计算机的输入部件。事实上，键盘也就是由这样一些按键开关组成。

在一次次按动开关时测量电阻上的电压变化并绘成曲线，将得到一串跳变的脉冲图象。这就是数字信号的波形。理想波形应是矩形脉冲，高低电平的跳变越陡越好，以尽量减少“非零非一”的不明确状态。实际上任何种类的开关电路都不可能没有过渡时间，所以脉冲的上升沿和下降沿都会在时间轴上倾斜，近似梯形波。

数字信号的传送有三种方式。例如，向计算机输入166这个数，可以采用以下方式：

1. 脉冲计数方式。





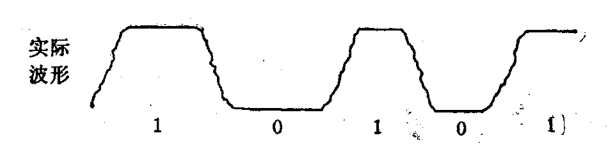


图 2-5 开关电路和数字信号

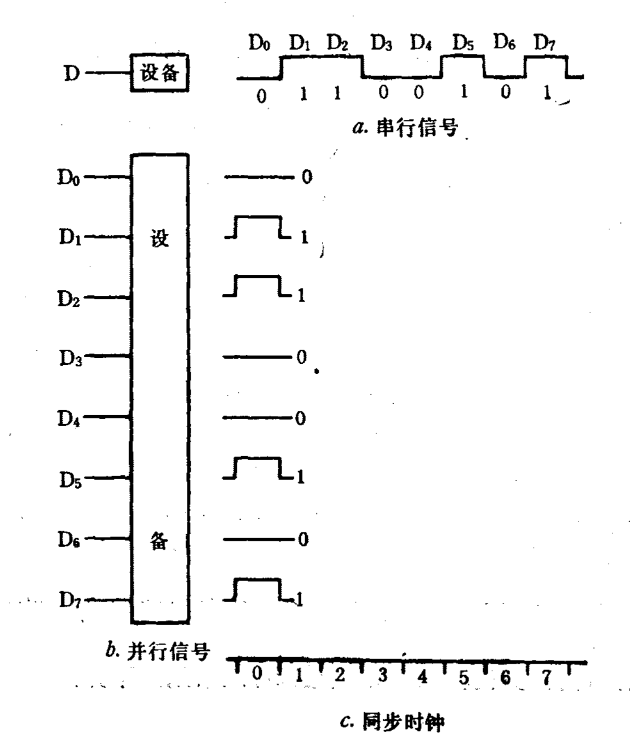


图 2-6 串行数据信号和并行数据信号

作为166个1来输入，即按动开关166次，机器以计数器接收。

2. 串行方式。

将数值按二进制数的形式表示(变成10100110)，将各位数码按一定的时间间隔顺序送入。只要收发双方的“时钟”是对好（同步)的，只需要8个时间单位和动作，便可将166送入(图2-6*a*)。

3. 并行方式。

用8个并列的开关分别同时发出上述各位数码信号，通过平行的电路传送。一个时间单位就可完成，速度快，但需要电路多(图2-6*b*)。

将按键开关用开关管取代，就成为电子开关电路。它不但能产生数字信号，它的开和关也必须用数字信号控制。因为要使管子可靠地导通或截止，基极或栅极需要输入幅度足够的高电平或低电平——也就是数字信号的1或0。所以，建立在开关电路基础上的功能电路都称为“数字电路”。

（三） 信息的数字化

需要计算机处理的信息远不止于0和1两种数码。数值信息中的正负符号、小数点，逻辑信息中的逻辑量，字符和图形，机器的指令和控制信号等等，在数字计算机里，都只能用它唯一“认识”的“1”和“0”来表示。也就是说，所有信息都必须数字化，变成二进制数形态的代码。

机器数的符号可以简单地用一位数码来充任，规定数的最高位为符号位，以0表示正号，以

1表示负号。但如参加运算的数都是正数，也可以不用符号，称为无符号数。11111111B作为有符号数等于-127，作为无符号数等于255。二者形态相同，有无符号，要事先约定。

小数点的表示法有两种。一种是定点法，例如规定小数点一律在最低位之后，机器里的数都是整数。若参加运算的数带有小数时，要先将各数按比例扩大成整数（相当于小数点右移），才能交给机器处理，运算后所得的结果，也要再按比例关系缩小才是应有的答案，这同笔算的方法是相似的。另一种是浮点法，一个数分成指数和尾数两部分表示，两部分都有符号。尾数是有效数码部分的定点数，指数则是指出小数点的位置（按符号规定由定点位置左移或右移的位数）。普及型微机的运算器不具备直接的浮点运算能力，只能用程序将它分解为若干定点运算步骤来完成。

字符的输入输出和处理，是计算机有别于计算器的一大特点。字符在机器内部也用二进制代码来表示，就像用点和横组成电码一样。国际上通用“美国标准信息交换码”，英文缩写是“ASCII”。它用七位二进制数来编码，共组成128个代码。其中00H～1FH用作控制码，20H～7FH是字符码，包括英文大小写字母和常用标点符号。我们在键盘上按下“A”键，最终是向机器内送入它的ASCII码——二进制数41H。

计算机处理的图形，必须分解为一个个的点，用二进制代码表示。如以0表示黑，以1表示白。如需要表示颜色和灰度，则用多位二进制数描述一个点。

0和1还是机器语言的“字母”。机器的指令也只能用它们组成。计算机中央处理单元一次输入输出的一组并行数字信号称为一个“字”，分为指令字和数据字两类。计算机的性能指标之一是“字长”。“八位”微处理器就是指字长是八位，一个指令字由8个“二进制字母”构成，可以有 $2^8=256$ 种不同的指令。当需要更多的指令时，也可以用连续两个指令字来表示一条

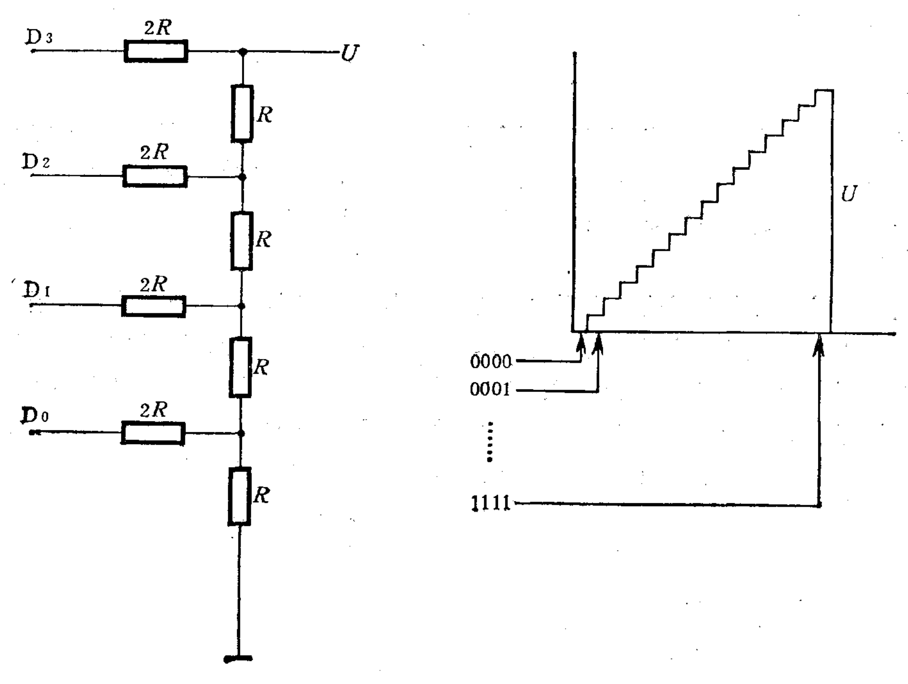


图2-7 数—模转换电路示例

指令。

通用数字计算机也能用于模拟量的处理，如作为工业控制机时，输入的常常是电流、电压等模拟量，需要在输入端用模-数(A/D)转换器件将它们变换为相应的数字量。A/D 转换的方法，是根据精度要求对模拟量分点采样，获得近似代表这个模拟量的一组离散的二进制数字信号，再行运算处理。实时控制所需的模拟量电信号，由计算机产生的数字信号经输出端的数-模(D/A)转换器件获得。

图 2-7 是一种最简单的D/A 电路——电阻梯形电路。由阻值R和$2R$组成的梯形网络，当数字信号 1 和 0 的电平差为nV时，输出电压$U = n\text{V} \times (D3/2 + D2/4 + D1/8 + D0/16)$。

可见，它实质上就是二进制数的“按权展开”。若 1 电平为 5V，输入数字信号 D = 0001B (1)时，$U = (5/16)$V；D = 0010B(2)时，$U = 2 \times (5/16)$V……D = 1111B(15) 时，$U = 15 \times (5/16)$V。输出电压同输入数据成比例。在输出端接上运算放大器(一种模拟电路)，即可获得所需的电压信号。例如将U放大16/5 倍，则可按伏特数值输出所需电压信号，如D = 1010B (10) 则$U = 10$V，等等。

计算机内各种信息都数字化，便于使用同样的元件、电路和信号，从而简化器件的基础结构——这正是复杂的机器系统所希望的。对各种形态相同的不同信息的区分，有赖于“事先约定”的原则，包括机器设计者的约定，使用者对机器的约定以及使用者对使用者的约定。如果不了解或不遵守这些约定，将会造成信息错乱。

三、逻辑电路

如果说元件、线路和电流是计算机的血肉，那么“逻辑”就是它的“灵魂”。在计算机技术术语中，“逻辑”一词无所不在，“总体逻辑”、“逻辑设备”、“并行逻辑”、“中断逻辑”……几乎把原理、关系、结构、方式、电路等等概念都包含在内了。这常使初学者感到困惑，计算机的“逻辑”究竟是什么?

逻辑学可分为形式逻辑、辩证逻辑和数理逻辑等分支。计算机技术所涉及的逻辑范畴，主要是数理逻辑领域的逻辑代数中的“二值逻辑代数”部分。逻辑代数是用数学方法研究思维规律的一种符号系统。而计算机正是模拟和协助人的思维的机器。所以它同逻辑代数的结合有其必然性。

数理逻辑虽然也采用数学的形式，但与普通数学不同，不是处理数量关系而是处理逻辑关系的。为了说明逻辑关系和数量关系的区别，举一个简单的例子。假设有三只电灯，它们的发光数量可用光通量表示，例如每盏 1000 流明。灯的亮和不亮这两种对立的状态，可用“逻辑量”表示，亮为 1，暗为 0。当它们用于阅读的照明时，开一只、两只或三只全开，发光量分别为 1000、2000 或 3000 流明，照明效果完全不同，应按需要的照度通过计算来控制开灯数，以取得最佳照明效果。这是一个数学问题。但如果是在洗印照片的暗室工作时间内(需保持黑暗)，精确计算光通量便失去了意义，开一只、二只或三只的效果相同(都会使相纸曝光失效)，即“1 + 1 + 1 = 1 + 1 = 1”。这是一种称为“或逻辑”的关系。在这种情况下，对开灯的控制便不是数学问题而是逻辑问题了。

电子数字计算机内部的每一个数字信号，都被作为一个逻辑量的电子符号。信号之间的关系，也都是逻辑关系。就连数值和算术运算，当它采用二进制形式时，每一位也是当作逻辑

量来处理的。整台计算机可以说就是一个规模庞大的组合逻辑系统。

(一) 逻辑代数

逻辑代数的内容很丰富，在计算机的应用仅涉及它的局部。二值逻辑代数有别于普通代数的特点有二。第一，它是一种二值系统，常数只有1和0两个，任何变量也只能有1和0两种取值；第二，逻辑常数或变量间的运算关系，只能是“与”、“或”、“非”三种。

计算机应用逻辑代数可分为三个阶段。设计阶段，将需要的各种信号间的逻辑关系用逻辑函数加以描述。制造阶段，以电路体现上述逻辑函数关系，用逻辑电路组成系统。运行阶段，逻辑电路按输入的逻辑量电信号“运算”产生表示逻辑函数值的电信号，从而实现设计的逻辑功能。

最基本的逻辑运算是：

1. 或(OR)运算。

运算符“+”或“∨”，逻辑表达式 $Y=A+B$。只要A或B任一个的值为1，则 $Y=1$；只有当所有变量值都为0时，$Y=0$。

2. 与(AND)运算。

运算符“·”或“∧”(字母间可省略)，表达式 $Y=A\cdot B$。当变量A与B的值都为1时，$Y=1$；若任一变量为0，则 $Y=0$。

3. 非(NOT)运算。

运算符是上置的“¯”，表达式 $Y=\overline{A}$。$A=0$ 时 $Y=1$，$A=1$ 时 $Y=0$。

以上三种逻辑运算的“真值”表见表2-1。

逻辑运算的优先顺序是：括号，非，与，或。

表2-1 基本逻辑函数真值表

AND			OR			NOT	
A	B	Y	A	B	Y	A	Y
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

从表中可以看出，逻辑或同算术加、逻辑与同算术乘很相似，仅在有无进位上不同。因而，它们分别也称为“逻辑加”和“逻辑乘”。任何逻辑关系都能用以上三种基本逻辑运算组成的表达式来加以描述。例如 $F=AB+BC+D$。式中的A、B、C、D是逻辑变量，F的值是由它们的取值所决定的，所以F是它们的逻辑函数。变量取不同值，函数值相应变化。

逻辑函数的“与或”表达式称为“积之和”，如 $F=A\overline{B}+C\overline{D}+BCE$。逻辑函数的“或与”表达式称为“和之积”，如 $F=(\overline{A}+B)(B+C)(A+D+\overline{E})$。任何一个逻辑函数都可以用“积之和”或“和之积”的形式表示。一个逻辑函数可以采用形式不同的表达式，也就是说一种逻辑关

系可以根据需要和条件选择不同的逻辑电路组合来实现，为逻辑设计带来灵活性。

逻辑代数有一个很重要的"范式定理"：n 个变量的任何一个逻辑函数，都可以展开成一组最小项的和或最大项的积，并且这种展开是唯一的。"最小项"是一个逻辑积——与项，其中每个变量都以原变量或反变量的形式作为一个因子出现并只出现一次。如三个逻辑变量A、B、C，有 $\bar{A}\bar{B}\bar{C}$，$\bar{A}\bar{B}C$，$\bar{A}B\bar{C}$，$\bar{A}BC$，$A\bar{B}\bar{C}$，$A\bar{B}C$，$AB\bar{C}$，ABC 这八个最小项。"最大项"则指每个变量都以原反变量出现一次并只出现一次的逻辑和——或项。如变量A、B、C，有 $\overline{A}+\overline{B}+\overline{C}$，$\overline{A}+\overline{B}+C$，…… $A+B+C$ 等八个最大项。在实用上，由最小项组成的"与或"表达式(也称"第一范式")用途最广，相应的"与或"逻辑电路具有实现各种逻辑函数的通用性，很多逻辑器件都建立在它的基础上。

根据上述规律，在已知一种逻辑关系时，就可以按其逻辑真值表写出函数的第一范式。例如，我们要设计一个电路，使它能比较两个逻辑变量是否不同(相"异")，若不同则函数值为1，相同则函数值为0。可按以下步骤进行：首先，按逻辑关系列出真值表，见表2-2；然后，写出每一行的最小项——变量取值为1的用原变量，取值为0的用反变量。最后用或运算符"+"号连接，得到表达式 $F=\bar{A}\bar{B}+\overline{A}B+A\overline{B}+AB$。

其中 $\bar{A}\bar{B}$ 和 AB 两项的真值为0，而或运算中的0项是可以省略的。所以实际上只需写出真值为1的那些行中变量的积之和，即 $F=\overline{A}B+A\overline{B}$。

表 2-2　XOR 函数真值表

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

将A和B取任意逻辑值进行验证，F的逻辑值均能正确表示它们间是否相异，符合既定的逻辑关系。这称为"异或(XOR)函数"，是常用逻辑函数之一，用运算符"⊕"表示。

一些多重的复杂逻辑式，有时需要展开成最小项的"与或"表达式，以减少逻辑电路的层次和延时。最小项表达式不一定就是最简形式，对给出的最小项表达式，有时又需要进行化简，以减少逻辑电路的元件数量。逻辑式的化简有多种方法，其中代数化简法同普通代数式的化简相似，是利用定律和公式进行变换和取代。逻辑代数有交换律、结合律、分配律、吸收律等基本定律，相应的公式见表2-3。读者可参照运用。

表 2-3　逻辑代数基本定律

定　律	基　本　公　式
交换律	$A+B=B+A$　　$A\cdot B=B\cdot A$
结合律	$A+(B+C)=(A+B)+C$　　$A\cdot(B\cdot C)=(A\cdot B)\cdot C$

续表

分 配 律	$A+B\cdot C=(A+B)\cdot(A+C)$　　$A\cdot(B+C)=A\cdot B+A\cdot C$
吸 收 律	$A+A\cdot B=A$　　$A\cdot(A+B)=A$
第二吸收律	$A+\overline{A}\cdot B=A+B$　　$A\cdot(\overline{A}+B)=A\cdot B$
反 演 律	$\overline{A+B}=\overline{A}\cdot\overline{B}$　　$\overline{A\cdot B}=\overline{A}+\overline{B}$
包 含 律	$A\cdot B+\overline{A}\cdot C+B\cdot C=A\cdot B+\overline{A}\cdot C$ $(A+B)\cdot(\overline{A}+C)\cdot(B+C)=(A+B)\cdot(\overline{A}+C)$
重 叠 律	$A+A=A$　　$A\cdot A=A$
互 补 律	$A+\overline{A}=1$　　$A\cdot\overline{A}=0$
0-1 律	$0+A=A$　　$1\cdot A=A$ $0\cdot A=0$　　$1+A=1$

（二） 基本逻辑电路

逻辑代数在电子数字计算机内的具体形式是开关代数。前面讨论过的开关电路，其输入和输出的数字信号正好表示 1 和 0 这两个逻辑量，因此可以作为构成逻辑电路的元件。逻辑电路也就是“逻辑开关”或“条件开关”，一般也称为“逻辑门”、“门电路”。

体现基本逻辑函数关系的电路可用最简单的按键开关组成，如图 2-8 中的 a 部分。这里，以开关动作表示自变量的逻辑值，按下为 1，放开为 0，灯泡中有无电流流过（亮或暗）则表示函数值为 1 或 0。显然，①a 两个开关串联构成“与”电路，只有两者同时按下（$1\cdot1$）时，灯泡才会亮（=1）。②a 两开关并联则构成“或”电路，任一开关闭合（如 $1+0$）灯泡都会亮（=1）。③a 的开关不按时灯亮，按下后形成旁路而灯灭，是“非”电路。

现在，我们以晶体管开关电路代替按键开关，成为图 2-8 中的 b 部分。与上面的灯泡电路不同，规定以电平而不是电流表示逻辑值，以加于基极的电平信号为输入逻辑量，输出电平信号由开关管集电极负载电阻 R 上取得。这时，图③b 就是一个最基本的晶体管开关电路。它采取共发射极接法，输出与输入的电压信号是反相的，负载电阻上的电流有无和 Y 点的电压高低在逻辑上也是相反的。输入低电平（0），管子截止，R 上无电流通过，Y 点电平与正电源一致，输出 1。输入高电平（1），开关管导通，集电极电流流过 R，产生电压降。Y 是很大的 R 值和很小的发射结电阻的分压点，所以接近地电位，输出 0。可见，最简单的电子开关电路也就是一个逻辑“非”门，或称“反相器”。这多少有点儿出人意料。当我们需要使输出和输入相同时，反而得用两个这样的开关串联起来，对信号两次反相才能实现。这是共发射极电平信号电路的特点。

同样，图① b 和② b 也就不是与门和或门，而相当于与、或之后又进行反相的与非门和或

非门。

实现常用逻辑函数的逻辑门电路可作为通用的逻辑元件使用，以构成具有复杂逻辑功能的组合逻辑和时序逻辑部件。常用的逻辑元件见表 2-4。本书采用国际通用的美国 MIL 标准的逻辑符号。

（三） 逻辑门 IC 的构造和电气性能

1．基本结构。

在集成电路中，与非门实际并不用图 2-8 那样的结构，而由一个特殊的多栅极 MOS 管或多发射极晶体管加上输出驱动器构成。图 2-9 是一个 PMOS 增强型四输入端与非门IC的平面图。

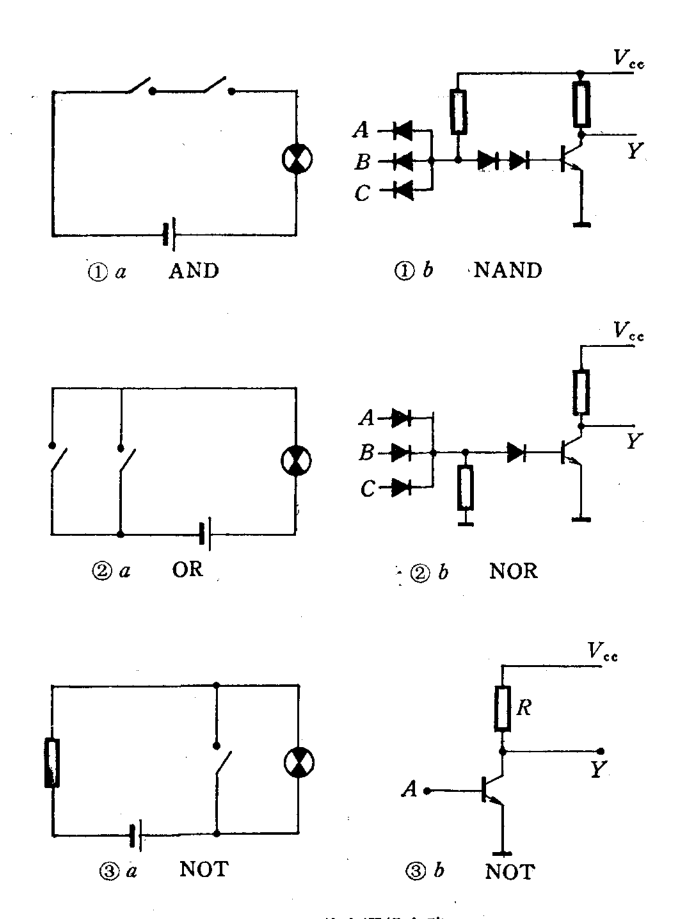


图 2-8 基本逻辑电路

表 2-4 常用逻辑元件

名 称	符 号	逻 辑 式	名 称	符 号	逻 辑 式
缓冲器 (BUFF)	A Y	Y＝A	与非门 (NAND)	A B Y	$Y=\overline{AB}$
非门 (NOT)	A Y	$Y=\overline{A}$	或非门 (NOR)	A B Y	$Y=\overline{A+B}$
与门 (AND)	A B Y	Y＝AB	异或门 (XOR)	A B Y	$Y=A\oplus B$
或门 (OR)	A B Y	Y＝A＋B	三态门 (3 State)	E A Y	E＝1时，Y＝A E＝0时，输出高阻态

说明 ①逻辑图中的小圆圈，表示信号反相。 ②“缓冲器”又称“放大器”，主要用于增加驱动负载的能力或电路的延时。 ③“三态门”详见下文。

它的左半部分是输入管，与反相器的区别仅在于栅极是多头式，并不比一个反相器占的面积更大。显然，各输入端全为高电平时，栅极为高电平，管子关断，D端输出为高电平；而任一输入端为低电平，就会在N区形成一条P沟道，使管子导通，输出低电平。它的右半部分是用一个MOS管担任的负载电阻。这个管子的栅极窄而长，并固定接低电平，形成的P型沟道相当于一个很大的电阻。

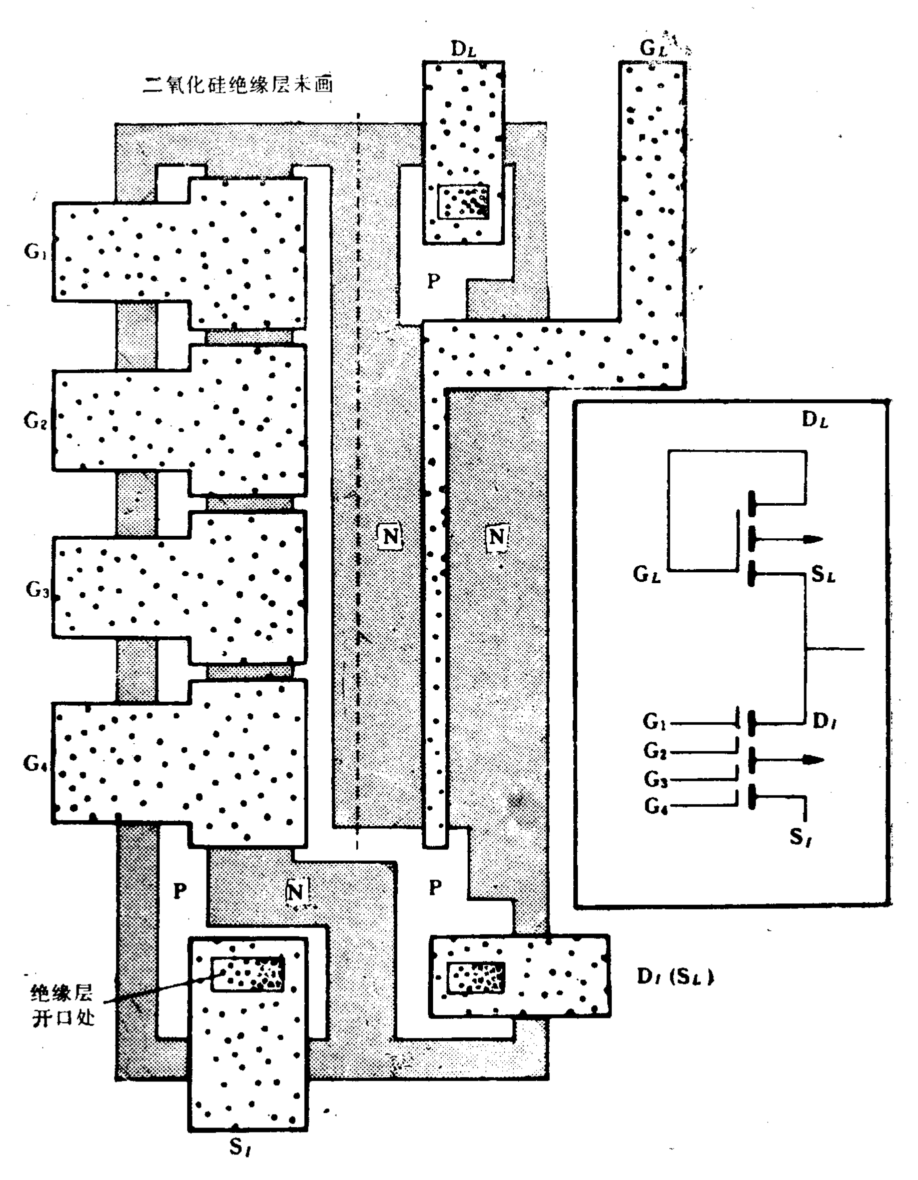


图 2-9　四输入 PMOS 型与非门结构平面图

双极型多输入与非门IC的基本结构见图 2-10。为了说明原理，先看图 2-8①b的二极管-晶体管逻辑(DTL)电路。当任一输入为低电平，相应的输入二极管获得正偏压导通，将A点电位拉低，晶体管发射结非正向偏置，处于截止状态，输出高电平。当输入端全为高电平时，

输入二极管都因反相偏置截止，A 点电平抬高，经基极串连的二极管给发射结加上正向偏压，管子导通，输出低电平。基极串联二极管的作用，是增加压降，使晶体管必须在 A 点电位高于 2.1V（三个 PN 结的压降）后方能导通，以提高对非数字电平的抗干扰能力。

图 2-10 中的多发射极晶体管是在 IC 的 P 型基区内制作几个并列的 N 型发射区硅岛，相当于输入二极管。它的集电结就相当于上述基极串联二极管。逻辑上完全等效，但结构简化，速度提高，功耗降低。这称为晶体管-晶体管逻辑（TTL）电路。多发射极晶体管在逻辑上是与电路，但由于它是“共基极”接法，当二极管使用，没有电流放大作用，集电极电流小，负载能力弱。因此它只宜作为逻辑门的与输入元件，后面至少应有一级共发射极接法的开关电路作为输出驱动器。最简单的就是加一个反相器，于是构成与非门。

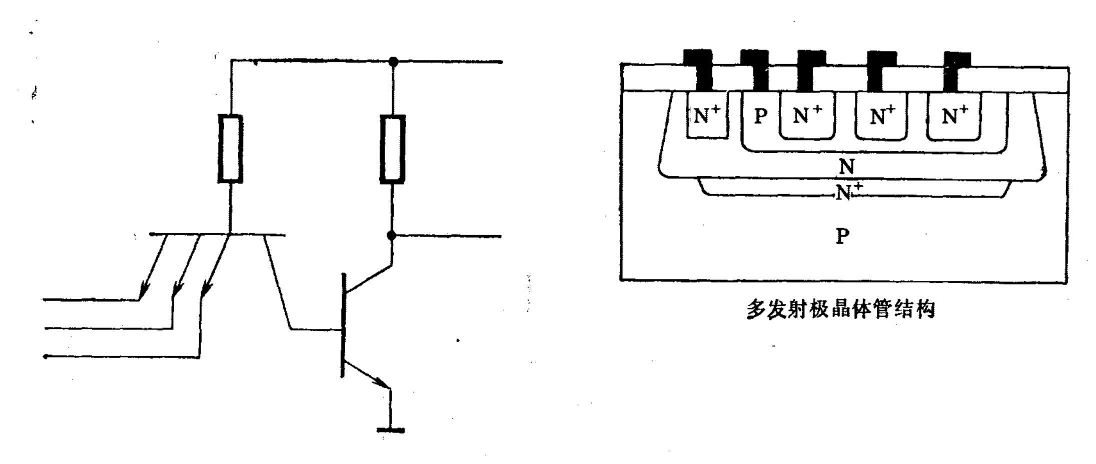


图 2-10　TTL 与非门

从电性能上看，与非门 IC 是一种最简结构。由于集成电路的高度繁复性，掩模工艺上总是要求基本图形的简洁一致。所以，与非门（以及或非门）被广泛用作组成各种逻辑器件的元件。

2. IC 电气特性和类型

根据开关元件的工艺和结构，逻辑电路 IC 有多种类型。它们可表示同样的逻辑，但在电气性能方面，如开关速度、功率消耗和驱动负载能力等方面有一定差别。

这里着重讲驱动负载问题。负载指逻辑电路的输出端所接的器件，多数情况是下一级逻辑电路的输入端。虽然逻辑电路属于电压信号系统，但电压和电流、电阻是不可分割的，服从欧姆定律。举个例子，一干电池的开路电压用万用表测量为 1.5V，将它接入电路加上负载后再量，电压会有不同程度下降。这是因为表电路的内阻远大于电池内阻，相当于一个阻值很大的负载，电流在全电路流过时压降几乎全部分配在表电路上。而接上阻值可和电池内阻相比拟的负载后，产生的分压作用将使部分压降落在内阻上，输出端电压随之降低。负载电阻越小，负载就越重，输出电压越低。要维持较高的输出电压，必须供给更大的电流。但电源的电流是有限度的，负载过重时将不能正常工作。

逻辑电路之间的关系与此相仿。前级可看作电源，后级的输入电阻大、电流小则负载轻，

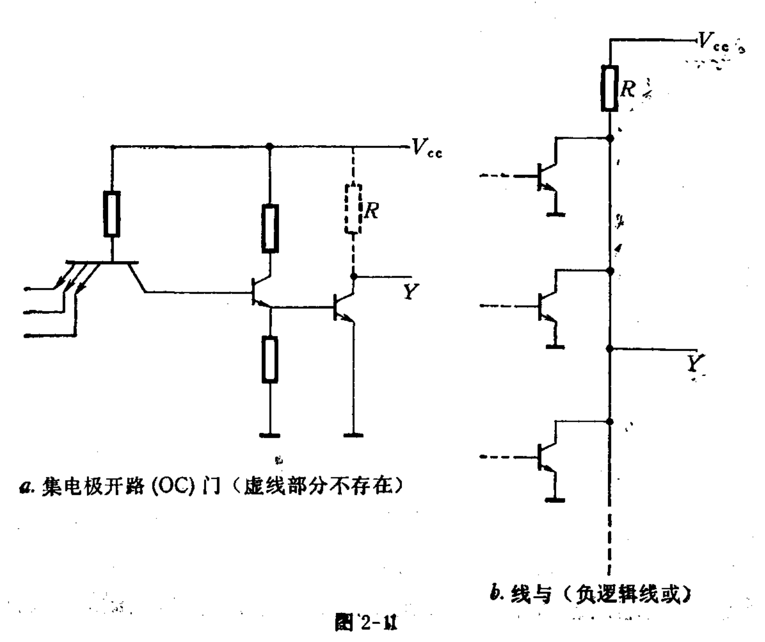
输入电阻小、电流大则负载重。一个输出端接上多个输入端时，因输入电阻并联阻值降低。负载过重，输出电平的幅度就不能达到数字信号的要求。TTL 电路的输入电阻是晶体管的发射结电阻，是一种较重的负载。一个 TTL 输出电流只能驱动不超出十个 TTL 电路。MOS 电路以绝缘状态的栅极输入，电阻极大，只需要微小的输入电流，一个 TTL 输出可供给几千个 MOS 电路的输入端。

除电阻性负载外，同时还存在着电容性负载。任何导体间都形成电容，除 IC 的结电容和内部分布电容外，当输出端接外电路时，宏观尺度的导线、器件的分布电容量是相当可观的。输出电平变化时，输出电流首先得使这些电容充电或放电，然后负载的输入电平才能变化。这就减慢了开关速度。电容充电需要时间由电容量和充电电阻决定，$T=2.2RC$。若 $R=1000\Omega$，$C=100\text{ppF}$，可造成 200nm 以上的延迟，接近于 Z80A 一个时钟周期的宽度，难于与它同步工作。

为了改善电路的速度、功耗和负载能力指标，IC 的结构，特别是输出驱动器部分，不断地有所改进，产生出特性不同的类型。下面先讨论 TTL 电路。

（1）无源上拉型（基本型）。图 2-10 中的与非门就是这种类型。它的输出端集电极接有一个上拉（负载）电阻 R。为了取得足够幅度的高电平信号，R 阻值在千欧级。当输入信号由高变低时，输出应由低变高。给负载电容充电的电流流经 R，由于 R 的阻力，使输出信号正跳变显著滞后于输入信号的负跳变，响应速度慢。输出变低时，负载电容通过导通的晶体管顺利地放电入地，不存在类似问题，但这时 R 上有电流流过，功耗较大。

这种电路性能虽较差，但有一种特殊用途——做成“集电极开路门（OC）”，见图2-11a，这个去掉 R 的电路部分是不能够独立工作的，需要外接集电极电阻。这种电路的特点是输出端可以多个并接在一起，共用一个集电极电阻（图 2-11b）。采取这种接法时，其中任何一个电路输出为低电平，就会把公共负载点 Y 拉成低电平，只有全部输出高电平 Y 才为高电平。这相当于各个门输出的“与”函数。简单的接线就起到一级与门的作用，故称为“线与”。这是



a. 集电极开路（OC）门（虚线部分不存在）

b. 线与（负逻辑线或）

图 2-11

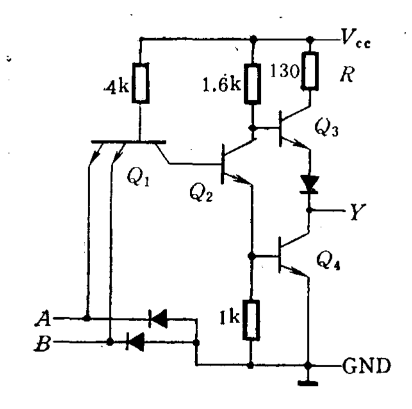


图 2-12　推拉式输出与非门（7400）

多个输出共用一条“总线”的一种方式，有利于节约器件，简化结构。

（2）有源上拉型（标准型）。图 2-12 的输出驱动器称为“图腾柱式”或“推拉式”。它的确很像新西兰毛利人部落村口那种图腾柱的样式。电阻 R 很小（≈100Ω）。Q_2是分相放大器，从发射极和集电极各输出一个互为反相的电压信号。当输入由高变低时，Q_2的反相输出变高，同相输出变低，导致 Q_3 导通、Q_4 截止，电源通过很小的电阻给电容充电，提高了 Y 点电平升高的速度。输出低电平时，因为 Q_3 截止，内阻很大，R 上基本无电流，减少了功耗。这种型式是现在用得最广的。但是，它不能做成集电极开路型。而直接将它们的输出端简单并接在一起是不允许的。因为如果各个输出的电平不同，则电流就可由电源正极经输出高电平者导通的Q_3，和输出低电平者导通的 Q_4，顺畅地流入电源负极，总阻值很小，电流会大到可将器件烧坏的程度。

图 2-13 是这种结构的变型，它的开关管采用“肖基特箝位晶体管”。因为普通晶体管由饱和过渡到截止时，排出基区多余的载流子需要的时间是影响开关速度的重要因素。肖基特箝位相当于在集电结上并联一个压降较小的“稳压管”（肖基特二极管），以阻止晶体管进入深度饱和。基区载流子减少，提高了开关速度。最常用的 74LS 系列通用集成电路便是这种类型。

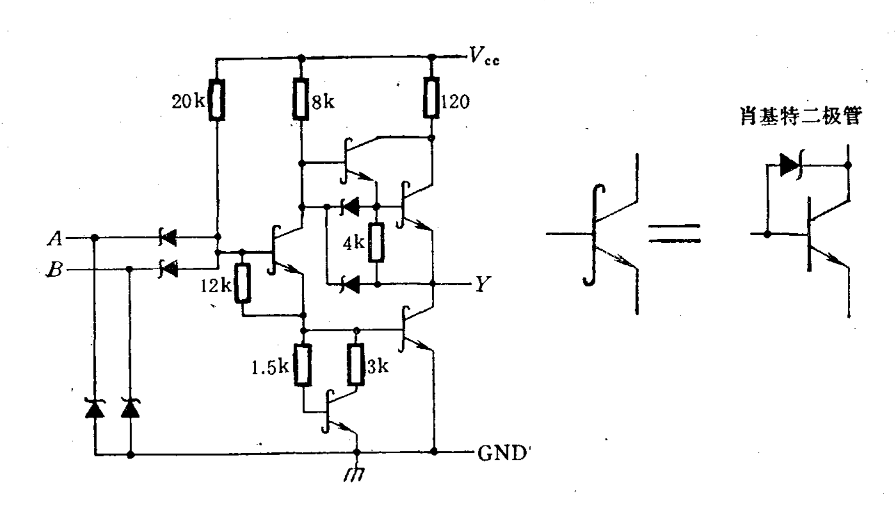


图 2-13　低功耗肖基特型与非门（74LS00）

（3）三态型。如图 2-14 所示，在标准型基础上增加了一个控制部分，使它成为一种独特的电路。当控制端 E 为高电平时，它对电路无影响，是一个标准的与非门，输出高电平或低电平。而如果控制信号变低，由于二极管 D_1 导通使 Q_1 截止，最终也使 Q_4 截止。而 D_2 导通使 D_3 也同时截止。于是输出端同输入端之间被两个高阻器件隔离，形同断路。这种状态是电气

的而非逻辑的，称为输出“高阻态”。这种驱动器就称“三态驱动器”。三态输出端是可以用线与方式并接在总线上的，但条件是任何时候都只允许一个逻辑输出，其余输出端都必须保持高阻态，这是逻辑设计应严格遵守的。

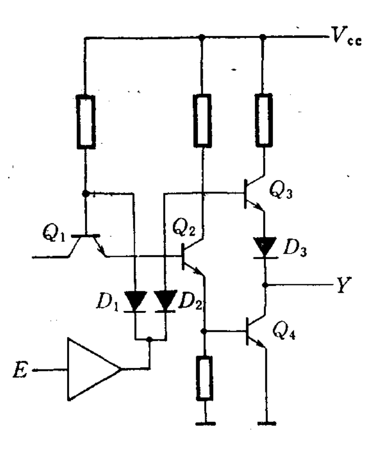


图 2-14 三态门

在总线方式中，三态输出门和集电极开路门适用于不同场合。当负载电容小或速度要求低时，使用后者结构较为简单。反之就应使用前者。需要驱动较重的外部器件时，一般都用三态输出驱动器。

MOS 电路也有相似的各种类型。普通型 MOS 也可做成漏极开路型，用于总线方式。相当于图腾柱型的是互补型 MOS(CMOS)。它以极性相反沟道型的 MOS 管代替漏极负载电阻，形成推拉结构，而且无需分相器(图 2-15)。它的功耗小、速度较快，得到广泛应用。通用集成电路中的 4000、74HC 等系列都是 CMOS 器件。由于它是两种载流子都参加工作，需要像双极型 IC 那样的隔离硅岛，因此集成密度较低。三态输出的 MOS 电路原理与双极型相似。

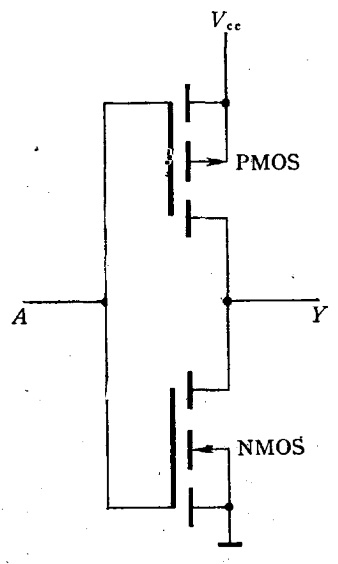


图 2-15 CMOS 反相器

（四） 正逻辑和负逻辑

前面提到的逻辑电平信号，都把高电平当作 1，低电平当作 0。这种规定是随意的。也可规定高电平作 0，低电平作 1。二者表达的逻辑意义并无区别。习惯上把前者称“正逻辑”，后者则相对地称为“负逻辑”。因为信号的电极性相反，在电路实现上是不相同的，最主要表现为“与或互换”。直观地看，将表2-1中 AND 函数和 OR 函数的真值中的“1”换成“0”，“0”换成“1”，它们就分别变成对方相应的真值表了。所以一个电路若对于正逻辑是与门，对于负逻辑，则是或门；同样道理，正逻辑或门，就是负逻辑与门。

常用逻辑电路输出是反相的。如用与非门实现与函数必须对输出信号再进行反相。在级联处理中，如果交替使用相反逻辑的器件，就可以减少反相环节。如一个与非门，在输入端采用正逻辑，输出端采用负逻辑，就相当于一个与门，只要下一级采用负逻辑的、带反相输出的器件，输出信号的逻辑类型又与最初的输入逻辑相同了。这样便能减少器件、功耗和延时。但逻辑关系复杂时容易搞错。

其次，负逻辑有利于采用总线结构。正逻辑的“线与”，对于负逻辑就是“线或”。接在总线上的任一输出为 1（低电平)时，总线的电平便与它一致，成为它占用的数据通路。

采用负逻辑还有利于降低功耗。因为器件输出高电平时流入负载的“源电流”小于输出低电平从负载流入的“吸收电流”。对于输出为 1 的时间远少于输出为 0 的器件，采用负逻辑使其经常处于高电平比较“省电”。例如微处理器的控制信号、芯片选择信号等，就常用负逻辑，以低电平为有效(1)。

四、逻辑部件

数字计算机系统的功能，除逻辑运算外，数值运算、数据存贮、信息传递和工作控制等也都建立在逻辑代数的基础之上，都用逻辑电路来实现。表 2-3 中那些基本逻辑电路，不仅能用来进行一种逻辑运算，还可有多方面的用途。例如，与门是最常用的控制门电路之一，异或门是算术运算的“半加器”，三态门是重要的接口电路，等等。它们作为逻辑元件，可以组成更为复杂的功能电路。例如，并行的两组三态门组成的总线收发器，两个与非门组成的存贮单元，若干或门组成的编码器……这些功能电路还可以进一步组合，构成具有更复杂、更完善功能的电路，直至集逻辑电路之大成的微处理器。具有各种独立功能的电路，便可作为组成计算机的逻辑部件。通用的逻辑部件都有集成电路产品出售。

下面主要讨论剖析时所要涉及的逻辑部件。一种逻辑电路可用作多种功能部件，以下只是大致的分类。

（一） 控制部件

控制部件的功能，是处理控制信号的发生、分配和转换，以实现对后级逻辑部件的选择和控制。

在控制系统中，以逻辑 1 表示有效信号，逻辑 0 表示无效信号。即正逻辑以高电平为 1，负逻辑以低电平为 1。当信号为低电平有效时，通常在信号名称上加横线标识，如$\overline{G}$，$\overline{R}$等。由于前面所说的原因，计算机控制信号常采用负逻辑。

1．基本逻辑门的控制功能。

（1）与门。用与门可作为控制一个信号有效性的条件开关（图 2-16*a*）。一个输入为主信号 A，另一个是控制信号。若控制信号 $G=0$，则主信号被封锁，输出信号无效，$Y=0$。当 $G=1$ 时，A 方能通过与门，$Y=A$。控制信号可有多个，其中任一为 0，$Y=0$；必须同时为 1，才能输出有效信号。若采用与非门，输出可变为负逻辑信号。

（2）或门。多个控制信号同时经一个或门输出，可合并成一个控制信号。其中任一个或多个输入有效，输出信号即有效（图 2-16*b*）。各输入信号为任意状态时，通过一个控制端送入逻辑 1，便可将输出信号强置为 1。

（3）非门。能使信号变反，与各种控制门配合，可用作正负逻辑的转换。如图 2-16 中，图 *a* 变为图 *c* 后，可控制负逻辑的有效信号并转换为正逻辑输出，$Y=\overline{A}G$。

（4）异或门。用异或门可控制一个信号以原状态或反状态输出。原反选择信号 $G=1$ 时，取反输出，$Y=\overline{A}$；$G=0$ 时，原值输出，$Y=A$（图 2-17 *d*）。原反控制对正负逻辑等效。

（5）与或门。见图 2-17 *e*。前级是若干与控制门，当任一组的主信号和控制信号都有效时，就能通过后级的或门输出。不同的控制信号组合，实现了对信号的条件选择。因为与或门的逻辑表达式是“积之和”的基本形式，可以用来表示一切函数关系，具有通用性，所以可构成各种逻辑功能电路。

2．译码器。

译码器是“与”控制的扩展。它由并列的若干与门（及与非门）组成。每个与门的输入是一个最小项。某个最小项值是 1，这个与门输出有效，其余输出均无效。如双 2-4 译码器74LS139（图2-17），每个译码器由四个与非门构成，有两个输入变量 A、B 和一个控制

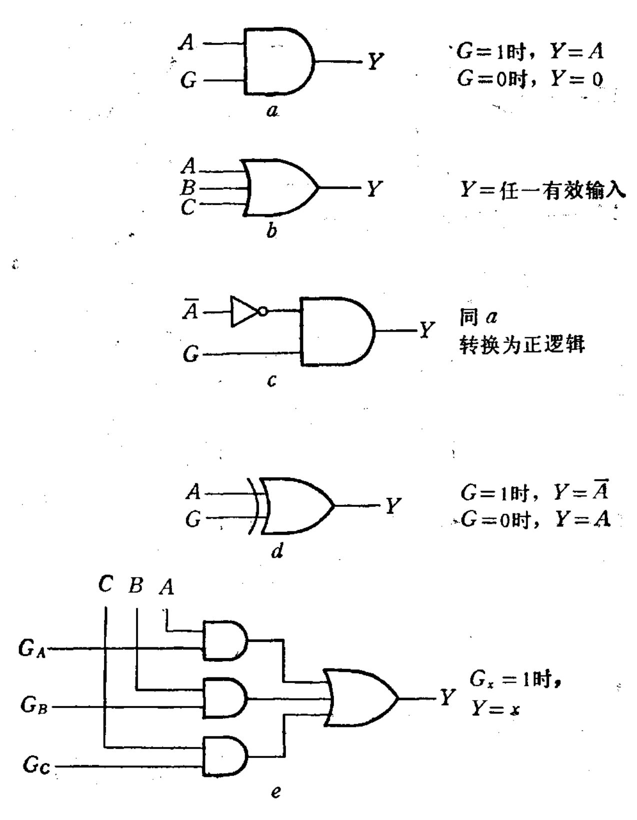


图 2-16 基本逻辑门的控制作用

信号G。对于每个输出信号，逻辑式为：$Y0=\bar{G}\bar{A}\bar{B}$，$Y1=\bar{G}A\bar{B}$，$Y2=\bar{G}\bar{A}B$，$Y3=\bar{G}AB$。G=1时译码被禁止，各输出端均为高电平（负逻辑的无效信号）。G=0时，按输入的A，B不同值，逻辑条件相符的与非门输出低电平（负逻辑1）；其它输出仍为逻辑0（表 2-5）。输出的4个控制信号可以各控制一个逻辑部件。若将它们编号为00B～11B（0～3），那么，

表 2-5 74LS139 逻辑功能

输入			输出			
启动 G	选择 B	选择 A	Y0	Y1	Y2	Y3
H	X	X	H	H	H	H
L	L	L	L	H	H	H
L	L	H	H	L	H	H
L	H	L	H	H	L	H
L	H	H	H	H	H	L

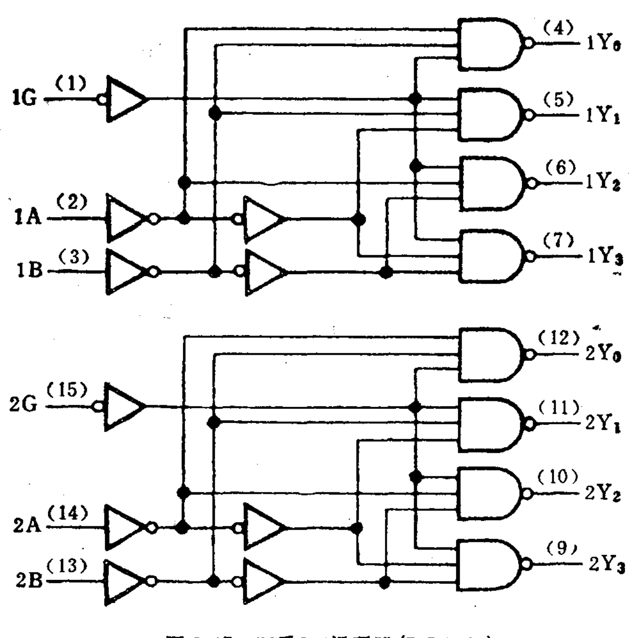


图 2-17　双重2-4译码器(74LS139)
(1-4数据分配器)

AB的数字信号代码，正好选择启动同号部件工作。

译码器在计算机中最广泛的用途是地址译码，将一组地址信号变成选通一个存贮单元或输入输出口的控制信号。

3。编码器。

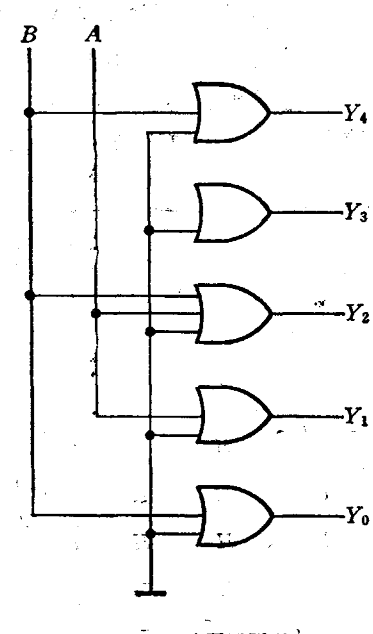


图 2-18　编码器原理

编码器是"或"控制的扩展。它由并列的若干或门组成，可由输入的一个有效信号产生一组既定编码的输出信号。图 2-18 的 5 个或门可产生一个 5 位的编码，具体的内容用连线的方法确定。如输入 A 时需产生的编码是 00110，就将 A 输入线连到 OR_1 和 OR_2 的输入端。B 的对应编码为 10101 时，将 B 输入连接到 OR_0, OR_2, OR_4 的输入端……。每个或门有一个输入端固定为逻辑 0（如接地），使这个门未被使用时输出逻辑 0 。这样，当输入 $AB=10$ 时，$Y=00110$B，$AB=01$ 时，$Y=10101$……。如果输入信号中不只一位有效，则输出是两组或多组编码相"或"的结果。如 $AB=11$，则 $Y=10111$B。

一般所说的"译码"，例如将一种代码"翻译"成另一种代码，需要用译码器和编码器联合实现。译码器按源代码产生一个有效信号，编码器根据这一信号输出特定的一组目标代码。

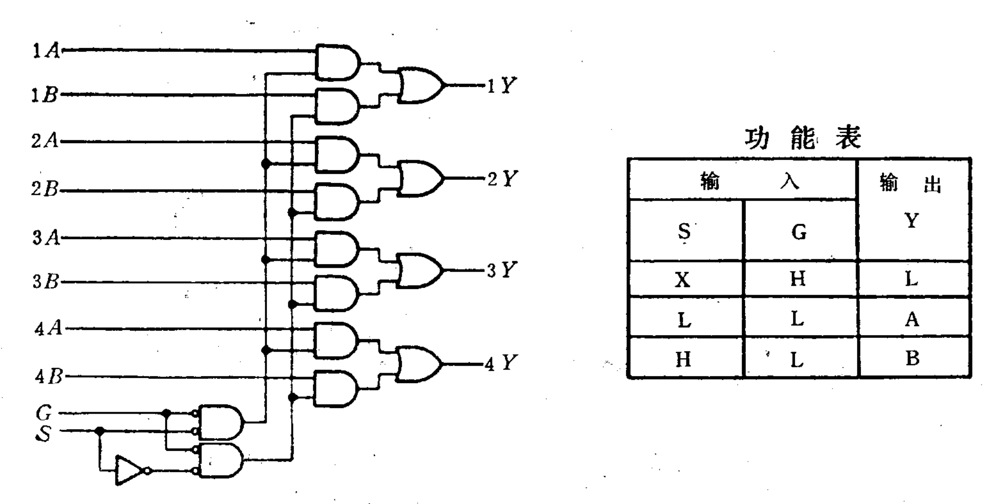
4。多路分配器。

多路分配器的功能是将一个输入信号按选择码送到

指定的一条输出通道。它的逻辑结构和译码器相同，是同一逻辑电路用于不同任务的一个实例。如图 2-17 的 2-4 译码器，G 可以看作待分配输出的源信号，而输入变量就成了选择码，由选择码 AB 的值决定将 G 送到哪一个输出端，逻辑关系同表 2-5。这时它是一个负逻辑的 1-4 多路分配器，无关位均输出无效信号（1）。

5. 多路选择器。

同多路分配器作用相反，多路选择器是在多个输入信号中按选择码选择一个输出。它是与或控制逻辑的具体运用。例如，2-1 多路选择器 74LS157（图 2-19），由四组与或门组成，可在两组 4 位输入信号中选择一组输出。当控制端 $\overline{G}$ 为高电平无效时，两组输入都被封锁，输出全为 0。当 $\overline{G}$ 为低电平有效，$S=0$ 时输出 A 组信号，$S=1$ 时输出 B 组信号。



功能表

输入		输出
S	G	Y
X	H	L
L	L	A
H	L	B

图 2-19 多路选择器 (74LS157)

（二） 接口部件

1. 缓冲器。

缓冲器的输出与输入在逻辑上相同。它的主要功能是在电气方面。用途之一是扩充负载。因为通常晶体管开关电路同时也是一个电流放大器，它的集电极输出电流远大于基极输入电流。当一个 TTL 电路的负载已近饱和尚不能满足需要时，可以在一路输出上增加一个缓冲器。缓冲器输出端便可扩充一组 TTL 负载。另外，有时还利用缓冲器来对信号传输延时，以协调各路信号的步调，避免逻辑门输入信号异步加载造成“竞争”和“险象”。

2. 总线缓冲（驱动）器。

由一组三态门组成，用作器件与公共的总线间的接口，以便将各输出端直接连接在总线上。例如八总线缓冲器 74LS244（图 2-20）有 8 个三态门，可用同一个允许信号控制。

3. 总线驱动/接收器。

又称总线收发器。两组方向相反的三态门并联在一条线路上，用互为反相的门控信号分别启闭，便可使信号朝某一方向通过，从而使输出输入能共用一条总线。如八总线收发器 74LS245（图 2-21）。

（三） 运算部件

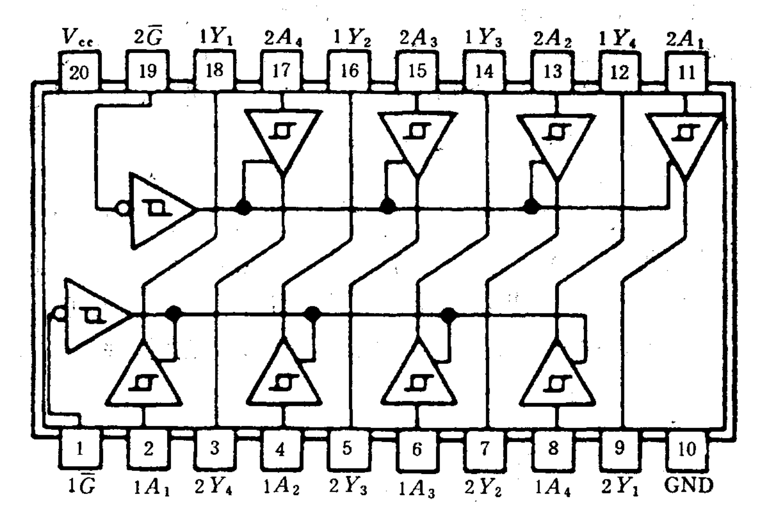


图 2-20　八位总线缓冲器(74LS244)

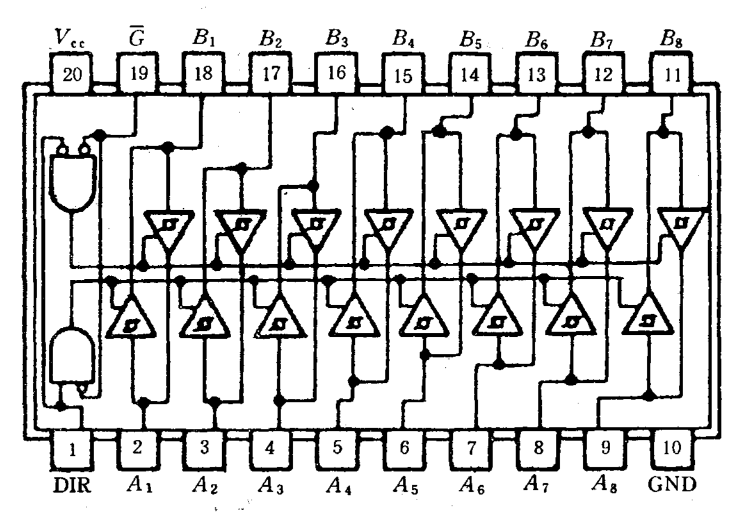


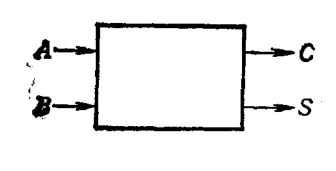
图 2-21　八位总线收发器(74LS245)

对于以逻辑电路构成的计算机，进行逻辑运算自是不在话下的了。需要讨论的是普通数学运算。机器运算在一定程度上是笔算的模拟，但它的两个基本前提——采用二进制和由逻辑电路完成，使它具有一些与十进制笔算不同的特点。

1. 加法器。

二进制位的加法只有四种情况：$0+0=0$　$0+1=1$　$1+0=1$　$1+1=0$ (并产生进位)。

因此，一个最起码的加法器，应有两个输入端 A 和 B，两个输出端 S (本位和)和 C(进位值)，框图如 2-22 所示。为了分析，将两个输出信号同输入信号的所有关系列表(表 2-6)，将会发现这两个表竟同“异或”逻辑和“与”逻辑的真值表完全相同，也就是说，加法运算的最基本关系可以看作逻辑关系。如果用这两种逻辑电路组合起来进行逻辑运算，其结果将同算术运算一致。这真是一种奇妙的现象：数值运算同逻辑运算看上去是风马牛不相及的两码事，由于二进制的媒介，居然汇合到一起了！



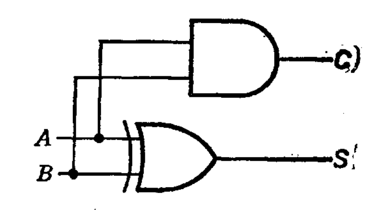


图 2-22 半加器

表 2-6 半加器真值表

A	B	S	A	B	C
0	0	0	0	0	0
0	1	1	0	1	0
1	0	1	1	0	0
1	1	0	1	1	1
同XOR运算			同AND运算		

于是，图 2－2 所示的电路便成了一个加法器，可用来作一位二进制数加法运算。但是多位加法器还不能简单地只由它组成，因为它的输入没有包括由低位来的进位值。因此它被称为“半加器”。用两个半加器可组成一个“全加器”。第二个半加器用来将本位和同低位来的进位值相加，两个半加器的进位通过或运算后作为向高位的进位输出（图 2－23）。

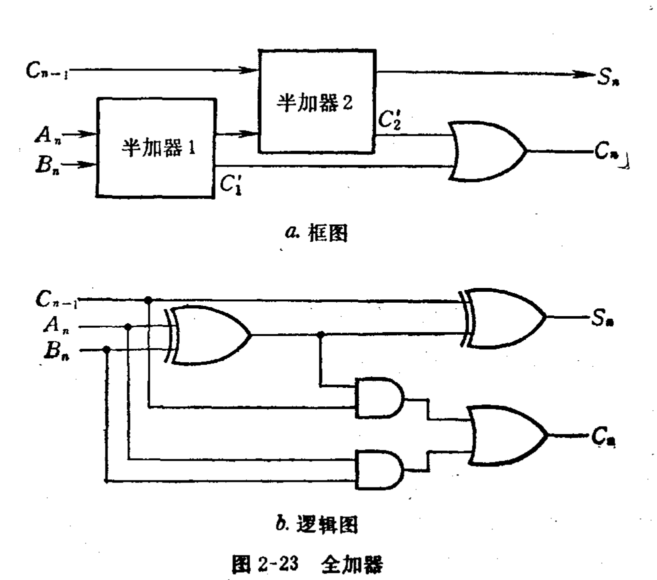


图 2-23 全加器

2．减法器。

仿照上述办法，用逻辑电路也可以组成一个减法器。但为了减化运算器结构，通常采取"以加代减"的方法，利用全加器来实现减法。将减数变为负数和被减数相加，与原数相减结果相同。在笔算里这是没有什么意义的，因为加负数仍旧要用减法。但由于机器算术采取了补数运算，就确实能够做到"以加代减"。

根据机器减法的需要，产生了"反码"和"补码"的编码体系。为了说明，先看看"补数"和"反数"的意义。

"补数"的概念对我们并不陌生，用整张钞票购物后的"找头"，就是应付货款的"补数"。如7是3对于基数10的补数，28是72对于基数100的补数，等等。由此可见，补数实际上是向高位借1减原数所得的差，所以加补数后再"还"掉高位所借的1，结果与减原数相同。例如：

$65-32=[65+(100-32)]-100=[65+68]-100=133-100=33$。

从笔算来看这简直是舍近求远，自找麻烦，要经过求补数(减)，加补数和减基数三步。但对于机器来说，求补和舍弃最高位都很简便，因此这种机器算法具有可行性。

减数的补数可用"反数"加1来求得。这种方法是商店售货员最熟悉的。对于1000元买764.25元货物后找多少的问题，他们会用心算先求出货价的十进制反数——同原数的每位数码都能凑成9的一个数235.74，然后在末位加1，成为补数235.75。这就是应给顾客的"找头"。二进制数求反数的方法，是对位凑1，也就是每位都同原数相反。例如1001B的反数是0110B。这很容易通过一组反相器实现，被减数和减数的反数相加，再在最低位加1，便实现了以加代减。因为运算器的位数是固定的，最高位向前的进位本来就可自然丢失，无需再减，所以，基本上全部用加法完成。

计算机的"原码"相当于上述的"原数"，但"反码"和"补码"的定义不完全同于"反数"和"补数"。它们是专为解决机器减法而设的，并要适用于加、减、正、负的不同情况，因而规定：

(1)原码、反码、补码都是有符号(正或负)数。

(2)原码加原码得到原码，反码加反码得到反码，补码加补码得到补码。

(3)正数和0的反码、补码都与原码相同(这是和反数、补数定义的主要区别)。

(4)负数的反码，是原码除符号位外各位取反；负数的补码是反码末位加1。

(5)反码的反码等于原码，补码的补码等于原码。

按上述规则，就可以用补码加法来进行减法运算。减数先将符号变反以变为加法。然后操作数都取补码形式，相加后最高位若有进位则丢弃，差数再次取补码。无论操作数是正或负数，都能得到正确的结果。例如：

$120-45=120+(-45)$,

$120=01111000B, 45=00101101B, -45=10101101B$,

$+120_{补}=0111000B, -45_{补}=1101001B$,

$+120_{补}+(-45_{补})=$ **1**01001011B $=+75_{补}=75$(阴影部分丢弃)。

因此，如图2-24所示的电路，在加法器的一组输入端增加了一列异或门作原反控制，就成为加/减法器。减数B的各位从原反门一端输入，减信号SUB加到每个门的另一端和全加器最低进位输入端。当SUB无效(0)时，B的各位以原码送入，最低进位输入也为0，此时为$A+B$运算。而当SUB有效(1)时，B的各位与1相异后均取反，成为$-B$的反码输入，并向最低位进1，等于加$-B$的补码。在加法器的输出端，最高位的进位值自然丢失，完成$A-B$运算，结果是

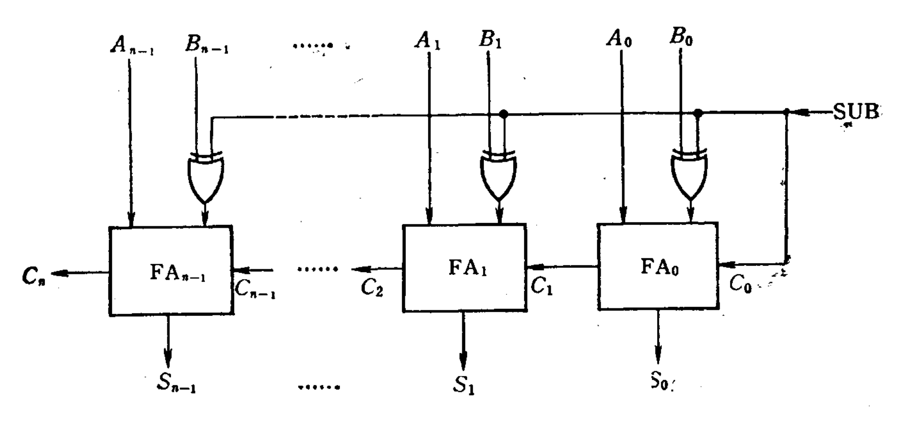


图 2-24　加/减法器

补码形式。

3．乘、除法器。

乘法的笔算方法，是用乘数由低到高的各位数字分别乘被乘数，得到的各个部分积则依次向左移一位，最后将所有部分的积加起来，得到结果。整个运算是由一位同多位数相乘、逐次移位和加法三个部分联合完成。

对于二进制数，因为被乘数乘以 0 等于 0，乘以 1 等于原数，所以上述过程的"乘"步骤变得极其简单。请看对 1101B×10011B 的笔算过程

```
       1101
   ×)10011
   ---------
       1101
      1101
 +)1101
 ----------
  11110111B
```

可以看出，各个部分积的有效数字与被乘数完全相同，只不过逐次向左移位而已(乘以 0 时只移位不产生部分积)。因而机器乘法可用加法和移位来完成。乘法器可用全加器、移位器和计数器、部分积寄存器等逻辑部件组成。

笔算除法的原理，也是一次次的乘法、减法运算和向右移位的组合，所以也可用加法器、移位器等构成除法器。但一般初级机都不具备乘、除法器的硬件，乘、除运算是用程序操纵加法器和移位器完成的。

4．综合性算术逻辑运算器。

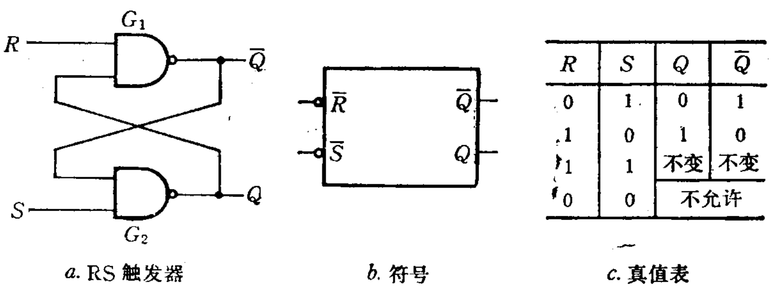
通用计算机需要具备多种逻辑运算和算术运算能力，若各用专门的运算部件，将会使系统变得相当庞杂。所以通常都采用综合性的运算器，称为"算术逻辑单元"(ALU)。

各种算术运算都可以用一定的逻辑运算来实现，这是综合性的算术逻辑单元能够成立的依据。根据逻辑代数的"范式定理"，任何逻辑函数都可以用由一个最小项构成的表达式来表示。因此，如果在一个含有输入变量全部最小项的"积之和"表达式中适当取舍，就能演变出各种算术逻辑运算的表达式来。根据这一原理，一个与或网络加上控制系统，就能构成算术逻辑单元。详细情况在"微处理器"有关章节讨论。

（四） **时序部件**

1．时序逻辑电路和触发器。

以上内容所涉及的逻辑电路，其输出的逻辑值只决定于当前输入的逻辑量（这里忽略了电路造成的传输延迟）。输入一旦改变，输出立即随之变化，前一输出状态便踪迹全无。作为智能机器的电子计算机，还需要有“记忆力”的部件。这种逻辑部件的输出信号能在输入改变后继续保持原状。即它的输出既可以同当前的输入有关，还可以同过去的输入有关。这就是“时序逻辑电路”。它的记忆元件仍然由基本的逻辑门构成。以上讨论过的静态的和开环式结构，是不具备这种能力的，必须采用闭环结构，利用信号的延迟和反馈，建立一种动态平衡的新型逻辑关系。



R	S	Q	$\overline{Q}$
0	1	0	1
1	0	1	0
1	1	不变	不变
0	0	不允许	

a. RS 触发器　　*b*. 符号　　*c*. 真值表

图 2-25　RS触发器

两个与非门按图 2-25中*a* 的方式连接起来，就能产生“记忆力”。G_1，G_2的输出 $\overline{Q}$ 和Q，分别作为对方的一个输入，构成反馈环路。置位端 S和复位端 R 都以低电平输入信号为有效。当R有效 S 无效时，按与非逻辑，$\overline{Q}=1$。反馈到G_2，使Q 端电平等于 0 。触发器进入复位状态。若一段时间后R变高无效，因为 $Q=0$ 的反馈，$\overline{Q}=1$ 不会变化，从而使 $Q=0$ 也得以保持，复位状态不变。也就是说，在Q端保存了$R=0$ 这一逻辑量。直到输入变为$S=0$，$R=1$，与复位过程相似，进入 $Q=1$，$\overline{Q}=0$ 的置位状态，在S无效后也能继续在Q端保存 1 。可见，输入的 R 或S低电平信号只起“触发”作用，在电路被触发而进入复位或置位两种稳定状态后便可撤消。这种电路由此得名为 RS 型“双稳态触发器”。它有两个互为反相的触发端，任何时候不允许 R和 S 的输入同时有效。那样将破坏了两者互反的逻辑定义。

触发器的工作对输入信号有比较严格的定时要求。如 S（R同样）电平由高变低后，经过与非门的传输延迟时间 T，Q 才变高，它反馈给 G_1 后又需经一个 T 的时延，$\overline{Q}$ 才变低。所以在 $\overline{Q}$ 变低以前，S 必须稳定不变，否则就不能达到所需的稳态。这就是说，当输入是脉冲信号时，触发脉冲的宽度必须大于 $2T$。

触发器的逻辑符号为一个方框，输入端有一小圆圈的表示低电平有效，否则为高电平有效。RS触发器是最基本的触发器。它需要一对输入信号，并且两个输入不能同为 0，这给很多实用场合带来不便。在它的基础上增加一些逻辑电路，可以构成具有不同触发特性的触发器，以适应各方面用途。图 2-26 中 *a*～*d* 是在 RS 触发器基础上的几种变型。

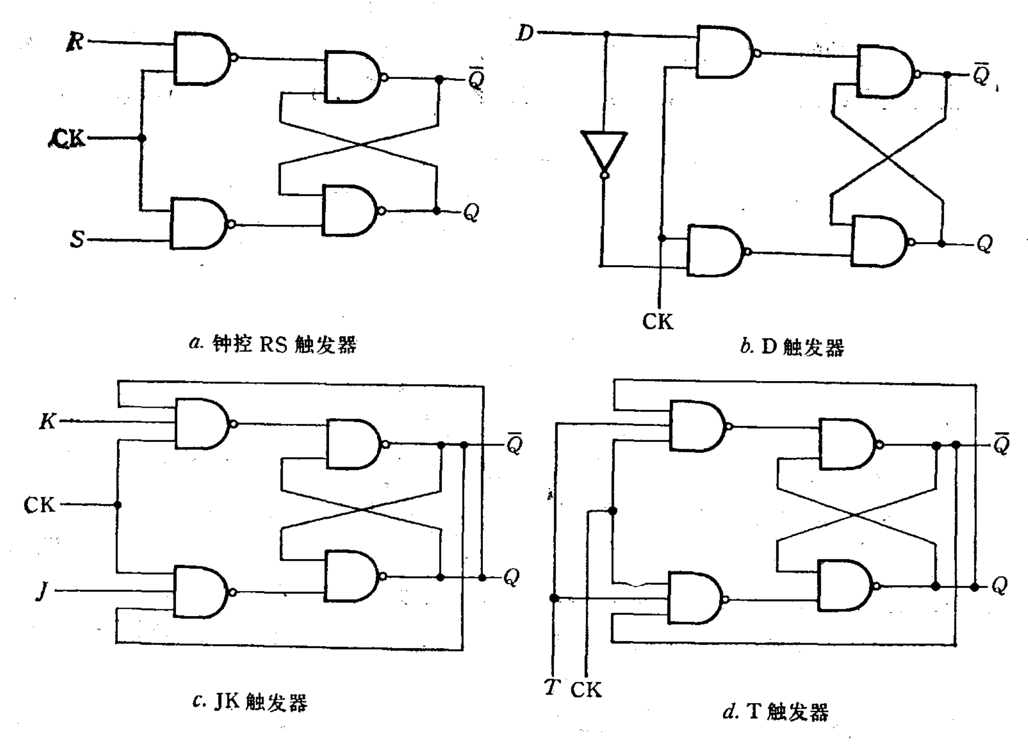


图 2-26　各种触发器

在图 2-26 中，a 是“钟控触发器”。它以一个控制信号通过两个与非门同时控制 R 和 S。这个控制信号按其通常的用途称为“时钟信号”。这时触发信号是否起作用得由时钟信号 CK 决定。CK = 1 时允许触发，CK = 0 时触发器被封锁，内容受到保护。这样就可以控制触发器定时工作。不过通常所说触发器的“时钟”并不一定就是定时信号，有时实际是“允许”、“启动”等控制信号。

在图 2-26 中，b 是“D 触发器”，它通过一个反相器使 RS 触发器变为单信号触发，在时钟高电平时，Q 端电平和 D 端输入一致，适于作为数据寄存器。

在图 2-26 中，c 是“JK 触发器”。将 a 中的 RS 触发器的两个输出信号引到相对的输入端，通过与非门控制触发信号 J 和 K。J、K 除与 R、S 有相同的逻辑意义外，还允许两个输入同为 1，这种情况下触发器由当前状态翻转到相反状态。

在图 2-26 中，d 是“T 触发器”。把 JK 触发器的 J、K 两端合而为一。显然，T 端为 0 时，触发器状态不变，T 为 1 时，将随每个 CK 脉冲翻转一次，因此，它可作为脉冲计数器。

实用的触发器大都采取钟控方式。但用正脉冲或负脉冲控制有一定缺陷。因为脉冲都有相当的宽度，即它总是“一段”时间。在这段时间里若输入端的数据有变化，触发器将因时钟继续有效而产生所谓“空翻”，把前面存入的正确内容冲掉了。为了使存贮可靠，时钟信号起作用的时间应尽可能短一些。而从其他方面考虑，时钟脉冲过窄是不利的。于是产生了“脉冲边沿触发”方式，即以时钟脉冲的上升沿或下降沿作为触发信号。

所谓“脉冲边沿”，就是逻辑电平跳变的过程(图 2-27)，是一个短时间范围。例如上升沿

触发方式，当时钟由低电平阈值经过渡期达到高电平阈值时，触发器立即动作，并随即进入相对稳定状态。此后时钟的高电平、下降沿和低电平，都不再具有触发和允许输入的作用，直到下一个上升沿的到来。因为脉冲边沿远窄于脉冲主体部分，所以可用作更为精密有效的定时。

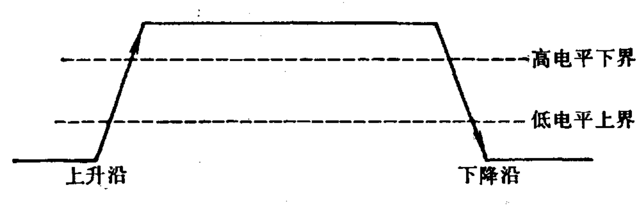


图 2-27　脉冲边沿

边沿触发方式可用"主辅结构"的触发器，即以触发器控制触发器的方法实现。下面以常用的双 D 触发器 74LS74 为例说明其工作原理。这种触发器逻辑图见图 2-28，可看作由三个钟控 RS 触发器组成。与非门 G_1G_2 组成主触发器，G_3G_5 和 G_4G_6 分别组成一对辅触发器。

清除信号$\overline{\mathrm{CLEAR}}$控制 G_1、G_4和G_5，预置信号$\overline{\mathrm{PRESET}}$控制$G_2$ 和G_6。它们都是低电平有效并互为反相，有效时无论时钟信号 CLOCK 和数据信号 D 是何状态，均能将主触发器复位或置位。在 $\overline{\mathrm{PRESET}}$ 和 $\overline{\mathrm{CLEAR}}$ 信号都无效时，数据 D 在时钟上升沿存入触发器，$Q=D$；其余时间 Q 保持不变。

分析比较复杂的触发器网络，须掌握几个要点：第一，抓住低电平输入信号这个"牛鼻子"，因为对于与非门来说低电平输入有"一票否决权"，必然将输出置为高电平；然后追踪信号的连锁反应，直至主触发器的 Q 和 $\overline{Q}$ 端。第二，当触发器某一输入由低变高时，由于门电路的信号延时，原状态的反馈信号将与新的输入共同发生作用，条件符合时可使触发器得以维持原状态不变，或者说触发器有一定的"惰性"。第三，辅触发器有时允许两个输出相同。

图 2-28 中 a～f 给出了触发器在复位、数据准备、触发、保持各阶段的示例，请注意比较各图间的信号变化。

a. $\overline{\mathrm{CLEAR}}=0$，迫使 G_1,G_4 都输出 1，使主触发器复位。b. 数据 $D(=1)$必须在时钟上升沿到来的一定时间之前出现在数据输入端，这称作"数据建立时间"，对于 LS 型最少 25nm。这个时间里数据已经进入触发器，使 G_5 和 G_6 输出改变，形成一种"预备姿式"，为下一步的变化作好了准备。因时钟低电平的制约，触发器其他部分不受影响。c. 当时钟脉冲上升达到逻辑 1 的电平，G_4因输入全 1 而输出 0，于是使 $Q=D=1$，$\overline{Q}=0$。时钟的正跳变犹如发令枪响，预备形态的输入数据便直奔终点。此刻要求输入端的 D 应有短暂的"数据保持时间"(LS 型≥ 5 nm)，使上述状态稳定下来。d. 在时钟正脉冲期间，若数据变化了($1\rightarrow 0$)，可使 G_5 的输出改变，但由于 G_4 输出的低电平反馈到 G_3 和 G_6，"否决"了 G_5 输出的影响，使主触发器状态不变。这两条反馈环路分别称为"置 0 阻塞线"和"置 1 维持线"。e. 时钟脉冲变低，实际上就是进入了下一个数据输入准备阶段，形成新的数据准备状态。f. 时钟脉冲上升沿将数据 0 置入主触发器。此后依靠 $G_5\rightarrow G_6$ 的"置 1 阻塞线"和 $G_3\rightarrow G_5$ 的"置 0 维持线"，在数据改变时保持原状态不变。这种触发器由此得名"维持阻塞触发器"。

各种类型触发器都可做成边沿触发器。边沿触发器在逻辑符号中加一"缺口"表示，加小圆圈的表示下降沿触发。

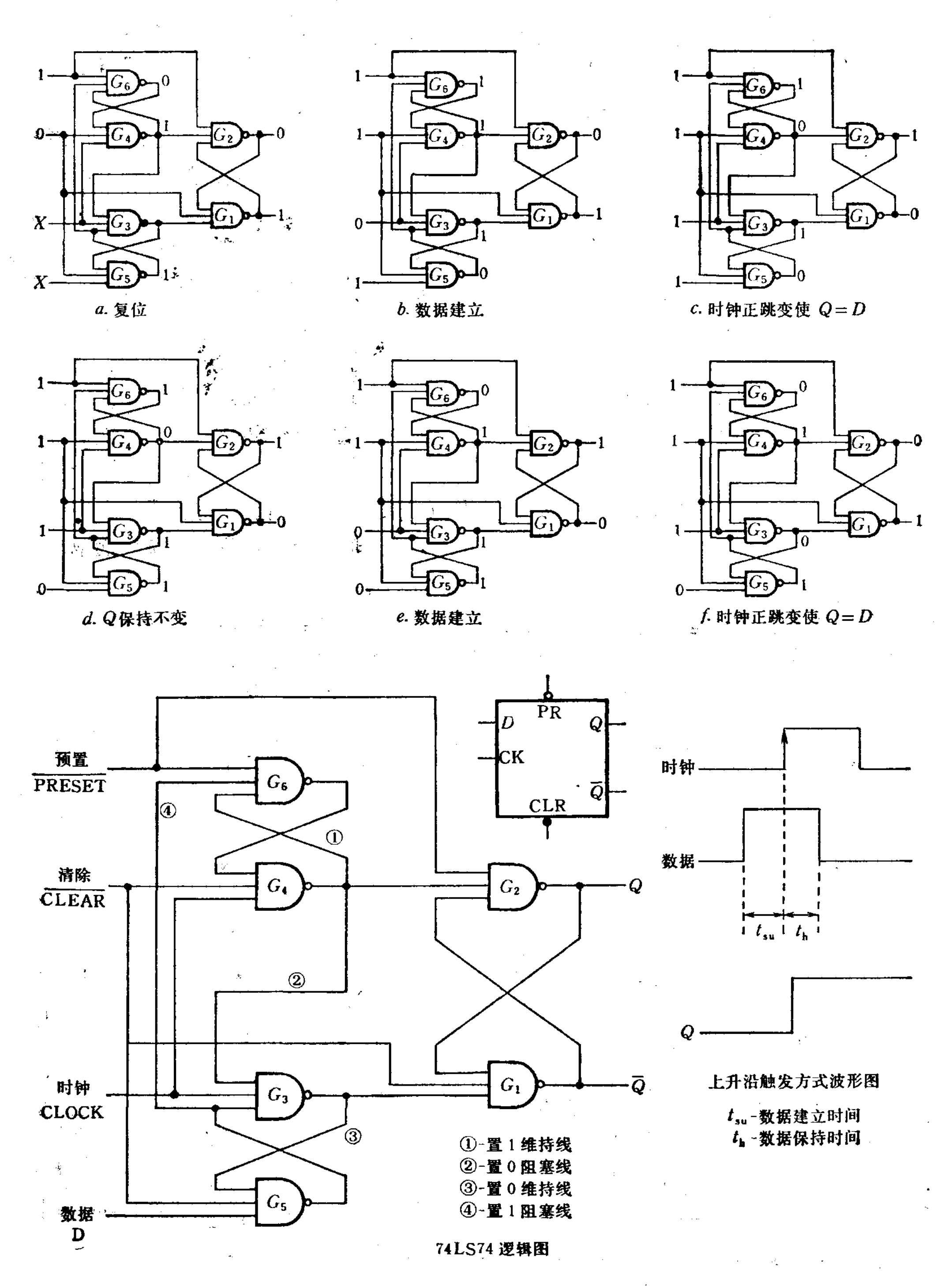
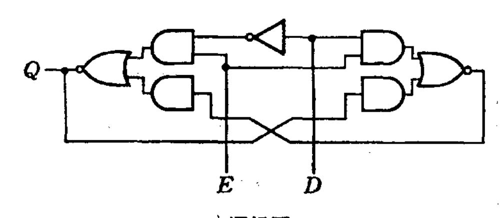
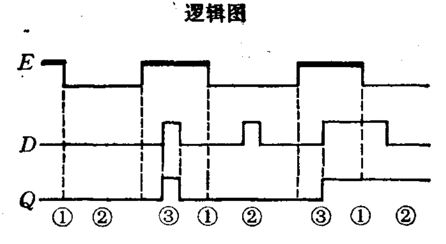


图 2-28 边沿触发方式的D触发器



逻辑图



①-E的下降沿，使 $Q=D$，锁存

②-E为低电平，保持 Q不变

③-E为高电平，Q随D而变化

图 2-29 锁存器(74LS77)

锁存器(又称"闩锁器")是和 D 触发器类似的一种电路，如 74LS77(图 2-29)。它具有与门和寄存器两种功能。当"允许"信号$E=1$时，电路相当于与门,$Q=D\cdot 1$，在此期间 Q 可随 D而变化。当 E 电平由高变低时，它将把下降沿之前的数据 D 锁存在 Q 端，在 $E=0$ 的条件下，电路成为寄存器，所存数据不受输入变化的影响。可见它和 74LS74 的触发特性有差别。不过，一般也常将 D 触发器称为"锁存器"。

2. 寄存器。

寄存器是计算机工作中暂时存放数据的器件。它可以用各种触发器加上控制输入输出的控制门等逻辑电路构成。每个触发器寄存数据的一位。各位触发器之间可有各种互联结构，形成不同特性和用途的寄存器。向寄存器存入数据就是将触发器置位或复位，而取出数据只是对输出端的电平进行采样，并不"取走"和改变触发器的状态。

(1)缓冲寄存器。是最简单的寄存器，可直接使用触发器，输入和输出端不加控制，输入改变后经触发器延时，输出随之改变，如同一个有记忆能力的缓冲器。

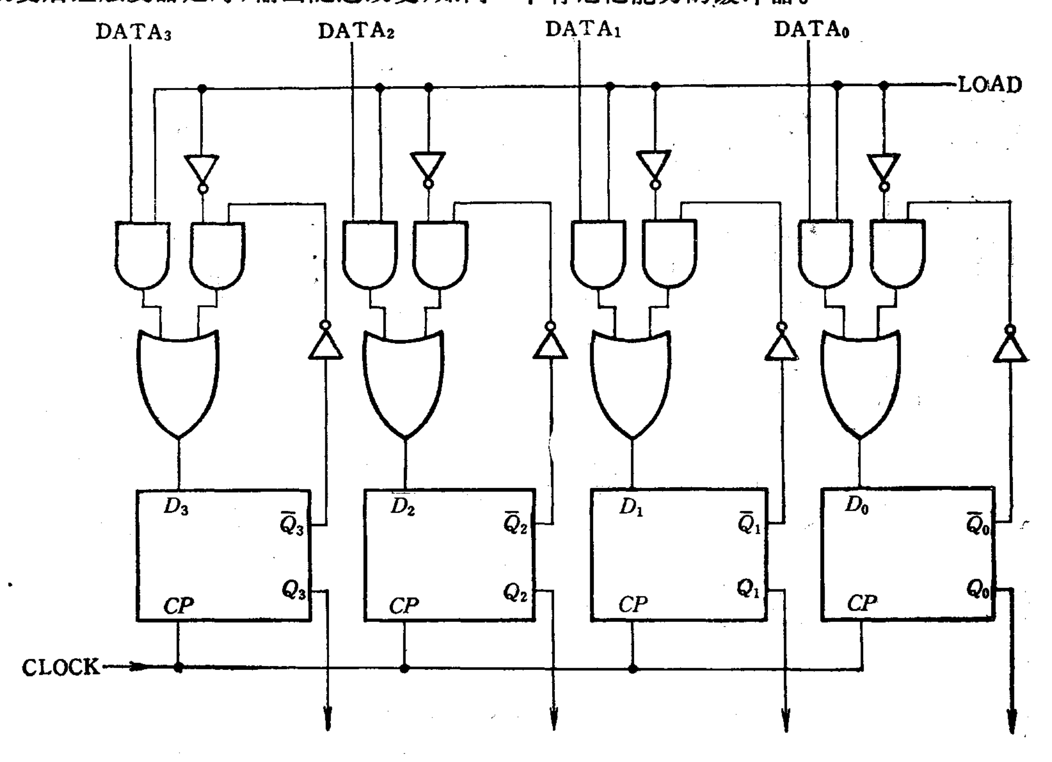


图 2-30 输入可控寄存器

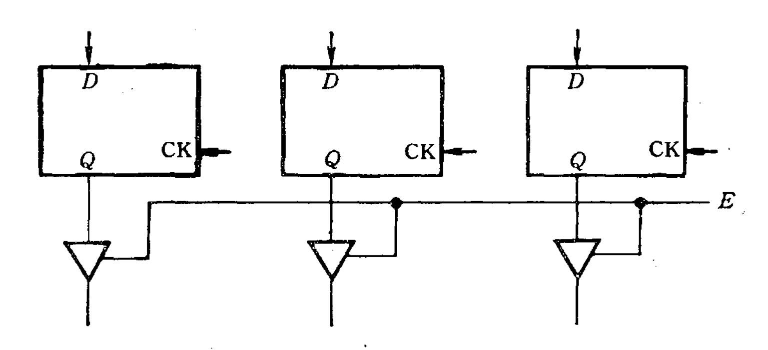


图 2-31　三态输出寄存器

（2）输入可控寄存器。虽然触发器可用时钟信号控制，但在同步方式计算机中，各部件使用同一时钟，这时它只起时标作用而不可能含有特定的控制信息。当触发器固定在数据线上的时候，为了能对出现在输入端的数据有所取舍，需要增加一些控制电路（见图 2-30）。当装入信号 LOAD 有效时，数据在时钟正脉冲存入。LOAD无效使输入数据被封锁，时钟触发仍将原存数据反馈存入，以保持不变。

（3）三态输出寄存器。将触发器输出信号经三态门输出，便可用于总线结构（图 2-31）。

（4）移位寄存器。移位是一种重要的数据处理手段，移位寄存器便是以时序逻辑实现数据移位的电路。一组触发器各输入、输出端首尾相连，一个触发器的输出信号作为相邻触发器的输入信号。在时钟脉冲的控制下，各触发器的内容便可按一定顺序传递。这就是最简单的移位寄存器。移位寄存器的左移和右移类型，由互连线路决定。实用的移位寄存器用若干门电路来控制信号在触发器间的流向，就可以实现左/右移可选择，串行或并行输入，串行或并行

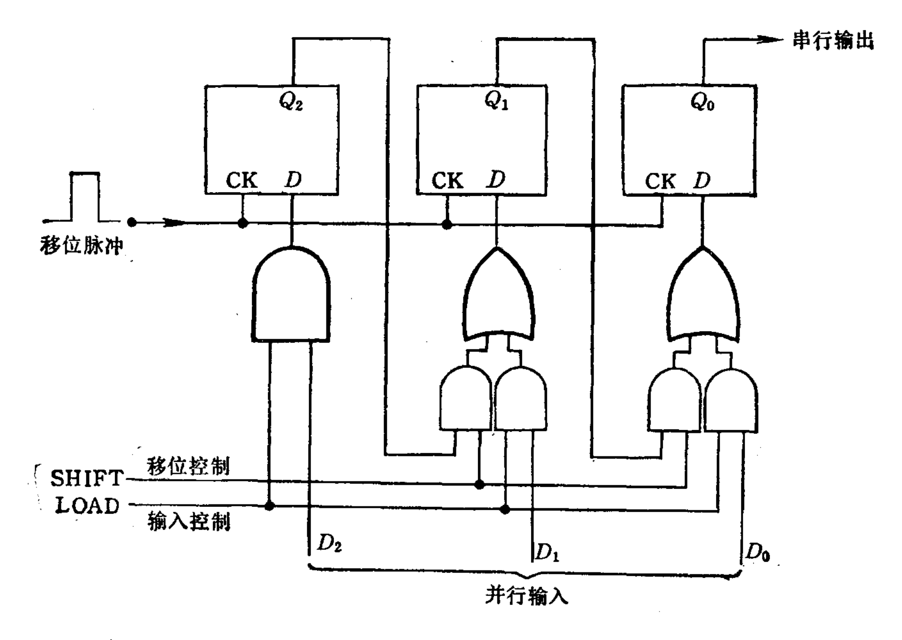


图 2-32　并入串出右移寄存器

输出等不同工作方式。图 2 -32 是一个由 D 触发器组成的三位"并入串出"右移寄存器。移位寄存器是一种用途广泛的逻辑器件。

3. 计数器。

触发器可以在外部脉冲触发下发生"翻转"的特性，是它可作为脉冲计数器的逻辑基础。因为每一次翻转相当于对触发器所存数据加 1：0→1 = 0 + 1，1→0 = 1 + 1（由 1 到 0 的电平跳变就可作为进位信号）。将多个触发器联结起来，加上预置（全 0 或全 1）、计数和进位等控制逻辑，便构成实用的计数器。

图 2 -33是用D 触发器构成的三位二进制计数器。每个触发器的数据 D 都来自本身反相输出端$\overline{Q}$的反馈，所以在每个时钟脉冲翻转一次。清除信号使各触发器都进入 0 态，准备计数。最低位触发器以需要计数的脉冲作为时钟信号，直接对它计数。以后各级都以前级触发器的$\overline{Q}$输出脉冲作为时钟脉冲，在前级触发器由 1 变 0 后（$\overline{Q}$ 输出正跳变）翻转，从而实现计数进位。各触发器 Q 输出端的数字信号组合，就是当前对输入脉冲的二进制计数值。

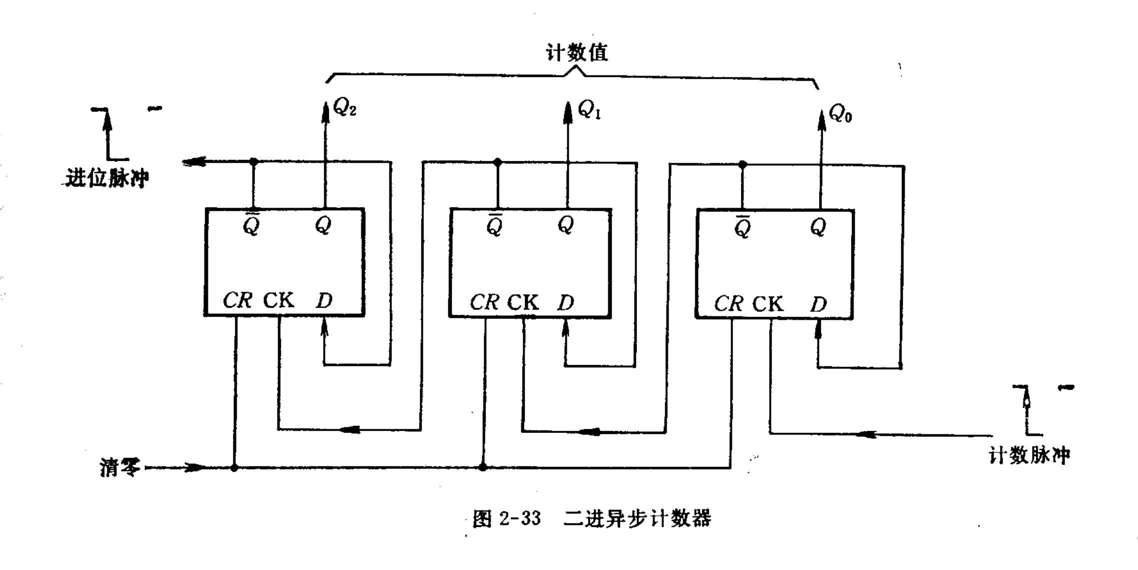


图 2-33　二进异步计数器

触发器的个数 n 决定了计数范围为 2^n，三位计数器最大计数111B = 7 。第 8 个计数脉冲到来时，计数值变成 000B。$\overline{Q_3}$ 输出的正脉冲是计数到 8 的进位信号。因此从整体来看，它可作为一个八进制计数器的一位。如果将它和一个二进制计数器级联起来，便成了 2 进 × 8 进 = 16进制计数器的一位。同样，十进制计数器用 2 进 × 5 进方式组成，如 74LS90 (图 2 -34)。它的五进部分有两个 T 触发器和一个 RS 触发器。T_B 接受$\overline{Q_D}$ 的反馈，在初始状态下，$\overline{Q_D}$ = 1，所以随计数脉冲（连成十进制计数器时就是二进部分的进位脉冲）翻转，Q_C 随 Q_B 进位脉冲翻转。它们计数到 011B 后，第四个计数脉冲到来时将同时发生两个事件：一是 RS 触发器的置位条件 $S \cdot \overline{R}$ 得到满足而使 Q_D 被置为 1；二是由于$\overline{Q_D}$原状态（1）的继续作用使 Q_B 和 Q_C 相继翻转为 0 。这就实现计数 100B。对于第五个计数脉冲，因为此时 $T_B = Q_D = 0$，T 触发器被禁止响应，RS 触发器则因 $S = Q_B \cdot Q_C = 0$，$R = Q_D = 1$，而被复位，于是，计数变为 000B，输出负跳变进位信号。

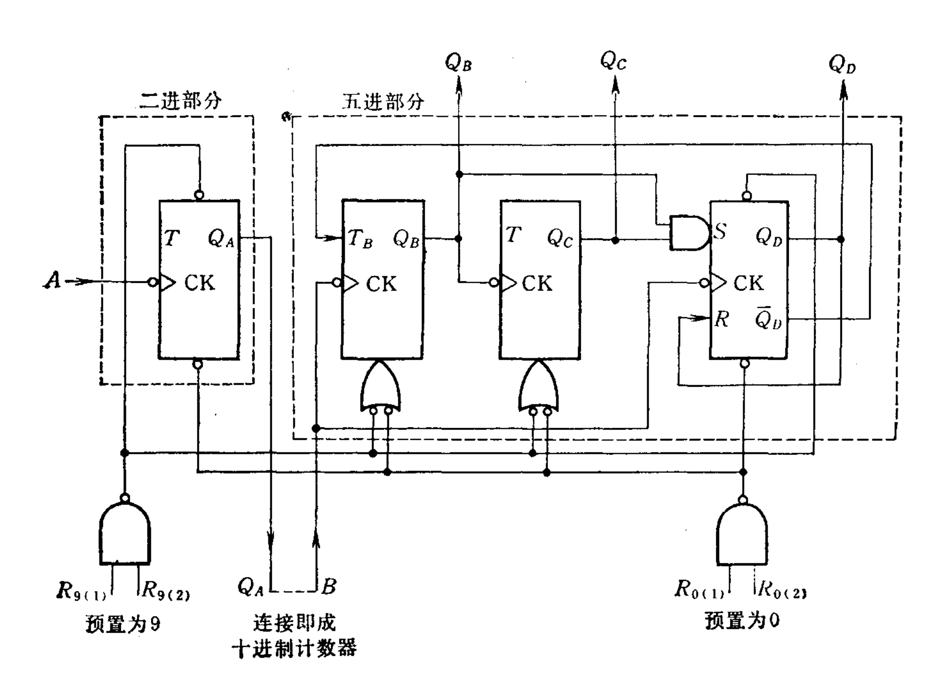


图 2-34　十进计数器(74LS90)

除加 1 计数器外,还有减 1 计数器和加 1 /减 1 的可逆计数器。

以上所举的计数器,触发器间的进位是由低位到高位次第发生的,例如当计数到 0111B 后再来一个脉冲,四个触发器依次翻转,成为 1000B。所以称为"异步计数器"。当计数器位数较多时，异步进位将造成很可观的延时。因而又产生了同步进位方式的计数器。同步计数器实质上是一个先行进位的加 1 运算器，用组合逻辑电路按计数器现值加 1 后将各位的结果存入触发器。它的工作速度快,但结构比较复杂,这里不作详细讨论。

计数器除用于本来意义上的脉冲计数外,还常用作分频器,在一种频率的脉冲信号基础上产生各种频率的脉冲信号。图 2 -35是图 2 -33 各触发器 Q 输出的波形图。对于二进计数器,输入两个脉冲才输出一个脉冲,所以 Q_0 的输出脉冲宽度是 CK 的两倍,频率则是 CK 的二分

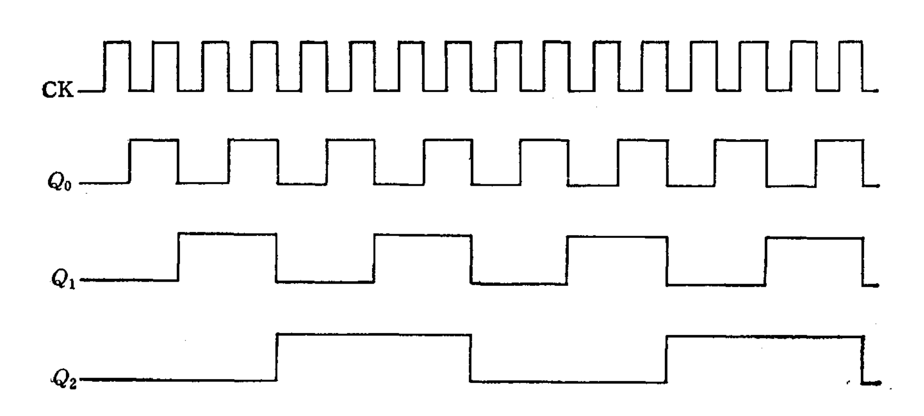


图 2-35　三位二进计数器(分频器)输出波形图

之一，可看作对输入频率进行了二分频。同理，四进计数器的输出 Q_1 是 CK 的四分频，八进计数器则是八分频器……。计算机里通常只有一个频率固定的时钟脉冲源，但各部分需要多种频率的脉冲信号，就要用各种组合的计数器来产生。

（五） 存贮部件

内存贮器是计算机的重要组成部分。因为存贮的数据量很大，它需要由大量的存贮元件构成。早期计算机使用的磁芯存贮器工作速度较低，后来已被半导体存贮器完全取代。寄存器就是一种半导体存贮部件，可用来组成存贮器。但它的结构比较复杂，难于满足大容量存贮器集成密度的要求，因而又开发出各种构造更为简单的存贮元件，形成各种类型和系列的存贮器 IC。

存贮元件应具有两种不同的稳定状态，以表示逻辑 1 和逻辑 0 。所谓“存贮”，并不一定就是把数字信号“装在里面”。一个逻辑元件，在向它输入 1 或 0 的逻辑量以后，能够稳定地保持特定状态，需要时能够在输出端给出原来的输入信号，便可作为存贮元件。因此最简单的开关电路都可作为存贮元件。图 2-5 所示的电路，若采用扳动开关，就具有存贮功能。数据用机械动作“存入”——将手柄扳到开（0）或关（1）的位置。因为弹簧的张力，开关状态能够固定不变，随时都可由 Y 点取出所存数据 0 或 1 的电平信号。

一个存贮元件只能存放一位二进制数。大量的存贮元件必须有序地组织起来，才能准确便利地进行存取操作。一般将存贮元件排列成二维或三维矩阵，并配以寻址、读写等外围电路。下面，以一个由简单开关所构成的存贮器模型(图 2-36)为例进行说明。

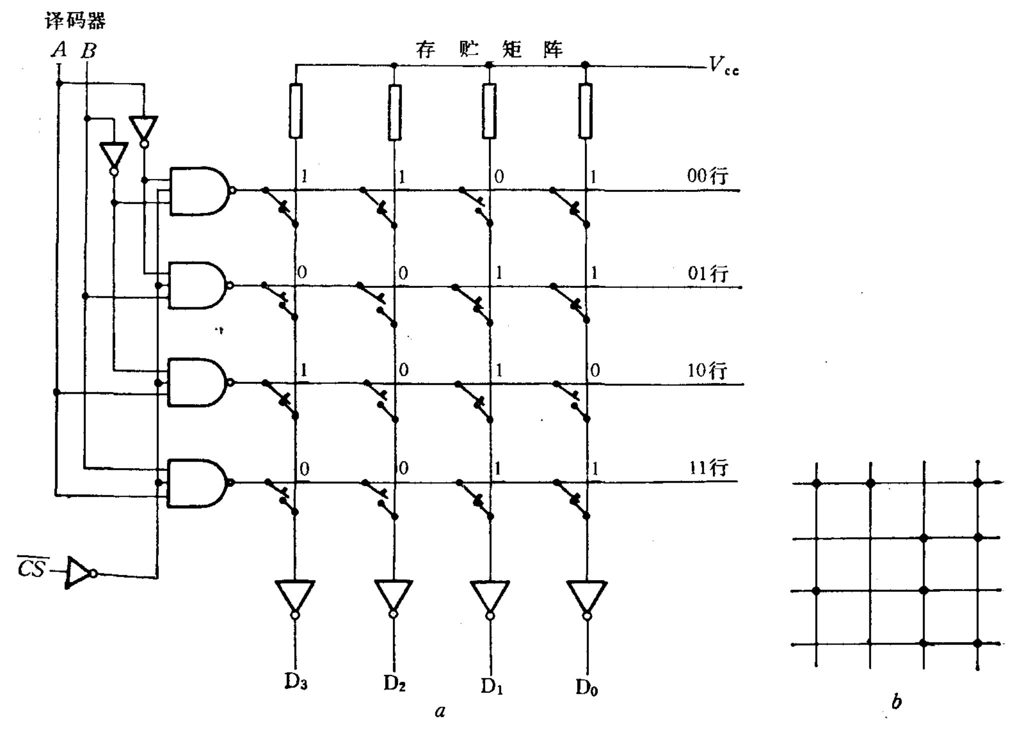


图 2-36 存贮矩阵模型

这个存贮器模型用16个开关排成4×4的二维结构。4条行线称为“字线”，4条列线称为“位线”，各通过一个非门输出一位数据。字线和位线互不相通。开关跨接在它们的交叉处。图2-36中b是a的简略表示法，用圆点表示交叉点上的开关闭合，其余都是断开的。这个存贮器可以存贮4个字长4位的数据，以开关的开状态存1，关状态存0。如图所示的存贮内容是：1101，0011，1010，和0011四个字，可以通过4个非门整字并行读出。

位线的一端通过电阻连接正电源，所以位线的基本状态为高电平。矩阵以一个2-4地址译码器作为字寻址机构。它的每个输出端连接一条字线。当本存贮器选择信号$\overline{CS}$无效时，译码器输出全1，字线全为高电平。所以无论开关通断，每条位线都是高电平，非门输出全为0。当$\overline{CS}$有效，存贮器得到一个字的地址信号，便会使一条字线进入低电平。线上的四个交叉点中，开关闭合处将把所联接的位线拉成低电平，从而使非门输出1。开关全部断开的位仍输出0。输出信号与存贮字的内容一致，实现了信息还原的要求（当然，这要求译码器输出端是集电极开路门）。

各种实用的存贮器的组织结构，逻辑上与此基本相同，只不过是用开关管取代了简单的开关，提高了工作性能而已，当我们不考虑器件的物理结构和特性时，常用图2-36b这种简洁的形式描述存贮器（主要是ROM）的逻辑结构。

1．只读存贮器（ROM）。

假若上述“开关存贮器”用在计算机里，显然不可能在机器运行中随时“扳动”以改变数据内容，必须预先设置好开关状态，在计算机工作时只能读取。这样的存贮器就称为“只读存贮器”（ROM）。实用的只读存贮器是在字线和位线间用开关管取代上述开关。

（1）掩模ROM。ROM可由晶体二极管、三极管或MOS管作为开关元件。图2-37是

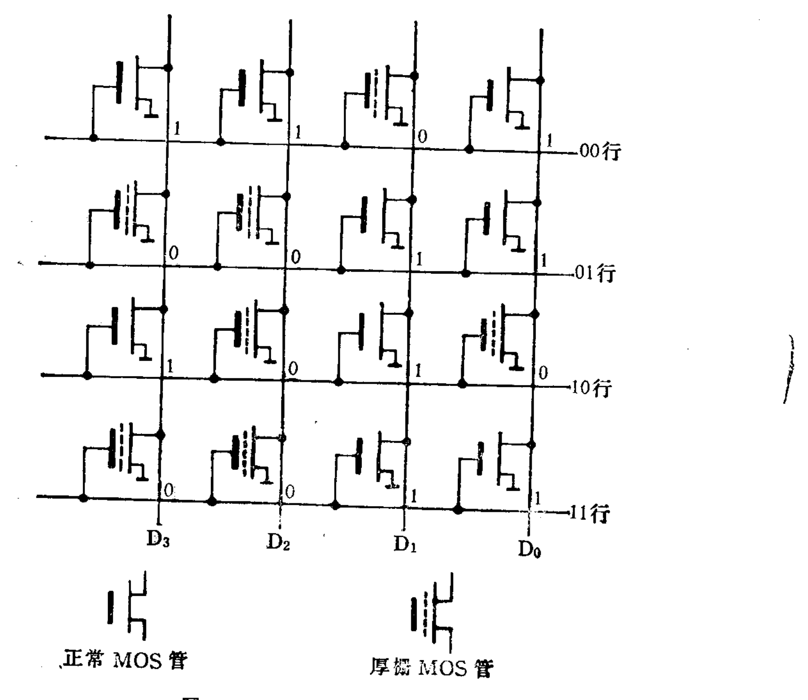


图2-37　掩模ROM矩阵

N 沟道增强型 MOS 组成的 ROM 示意图。矩阵结构和所存数据与前述模型相同。字线连接栅极，位线连接漏极。因为增强型 NMOS 管输入高电平时开启，译码器输出的有效信号应当为高电平。阵列里的 MOS 管有两种。一种是正常的管子，当栅极加高电平时管子导通，漏极输出低电平。另一种管子栅极下面的 P 区空穴浓度远较正常为高，即使加上正常栅偏压也不能导通，等于一个断开的开关，漏极输出高电平。这些存贮元件实际上是作为漏极开路的反相器，因而各个字的同一位可采取"线与"方式通过一条位线输出。

为什么不让开关断开的部位保持断路，而要多此一举地做一个"不通"的管子呢？这是由生产和经济的需要决定的。各种 ROM 存贮的数据不同，若专门为某一特定需要设计制造，因为批量小，成本将高得不能接受。因而产生了两阶段生产方式。首先按一定容量规模在基片上制作完全相同的 MOS 管阵列("全 0 ROM")，但只到最后的表面氧化工序为止。这种半成品芯片设计和工艺简单，可大批量生产，单片成本很低。需要固化软件的厂家购入这种半成品后，按存贮的数据设计出一种简单的掩模图样，只在存 1 的管子栅极位置留出窗口，通过光刻将这里的二氧化硅层刻蚀掉，然后进行增强离子注入，使窗口下空穴浓度增高。最后的开口、布线工序所用的掩模又是整齐划一的。不需专门设计。这样制成的 ROM，其内容完全由第一张掩模的图形决定，所以称为"掩模 ROM"。这样分段制造的费用显然比全程特制低得多，不过仍要求具有一定批量，它适合用来固化成熟的、推广应用的系统程序和数据库等。LASER 301 的系统程序 V2.0 便固化在一片掩模 ROM——HN613128 中。

(2)可擦型可编程只读存贮器(EPROM)。掩模型ROM适于永久性数据的存贮，对它的内容是不能修改的；要修改只能重制新片。为了满足用户和小型单位自己固化程序和数据，以及必要时还可对固化的内容修改完善的需要，又研制出可用电气手段写入信息的各种可重写只读存贮器(PROM)。其中，可用紫外线"擦除"全部存贮信息并可重写的 EPROM 得到广泛应用，27 系列 IC 便是常用的 EPROM。

EPROM 的基本单元是叠栅 MOS 管，结构见图 2-38。这是一个增强型 NMOS 管，只不过它的栅极是重叠的两层。上层是多晶硅控制栅，它下面二氧化硅绝缘层中还封闭着一片硅

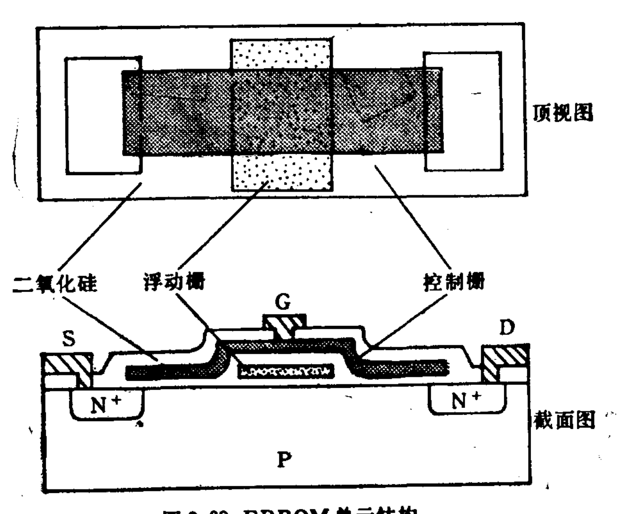


图 2-38 EPROM单元结构

“浮动栅”。所有叠栅管在初始状态时与正常的 MOS 管无异，在控制栅加上正偏压即可开启，相当于全部存贮着 0（通过反相驱动器输出全 1）。存入数据时，只改变需存 1 的管子的状态，方法是用地址信号选通目标单元，在源极或漏极同衬底间加上远较工作电压为高的反向电压（25V～30V），使 PN 结“雪崩击穿”。能量很高的电子穿越PN结，其中一部分有可能穿透极薄的二氧化硅层注入浮动栅。这一过程持续一定时间后，浮动栅里积累起相当多的电子，带负电荷。由于绝缘层的阻挡，电子很难扩散消失。通过静电感应，使浮栅下面 P 区的空穴浓度增高。这样，即使控制栅加上正常偏压也不能形成 N 沟道，管子保持关断状态，相当于写入了1。

EPROM 里的信息可以保存数年以上，直到浮动栅里的电子缓慢地“挥发”掉。存贮的信息需要擦除时，用一定强度的紫外线通过管壳的石英窗口照射芯片，浮动栅里的电子获得能量，二氧化硅绝缘电阻降低，电子便能扩散掉，所有的管子都恢复到初始态，可供重新写入。用户只要配备简易的“EPROM 写入器”、“EPROM 擦除器”，便能方便地固化自己的程序，甚至进行小批量生产。现在的很多游戏机节目卡就是用 EPROM 复制的。

从上述 ROM 的结构不难看出，它们实质上是一个与门阵列和一个或门阵列的组合逻辑电路。与阵列构成译码器，存贮矩阵就是或阵列（线或）的编码器。所以按照“范式定理”它能用来实现任何逻辑函数。只要把真值表存在编码矩阵中，自变量送入译码器，或输出便是函数值。

2. 随机读写存贮器（RAM）。

计算机里需要更多的是既能读出又能随时写入数据的存贮器件，即随机读写存贮器。它们是内存的主要部分，从结构和性能上可分为“静态 RAM”和“动态 RAM”两类。

（1）静态 RAM（SRAM）。这种存贮器的单元电路以双稳态触发器为基础。图 2-39 是

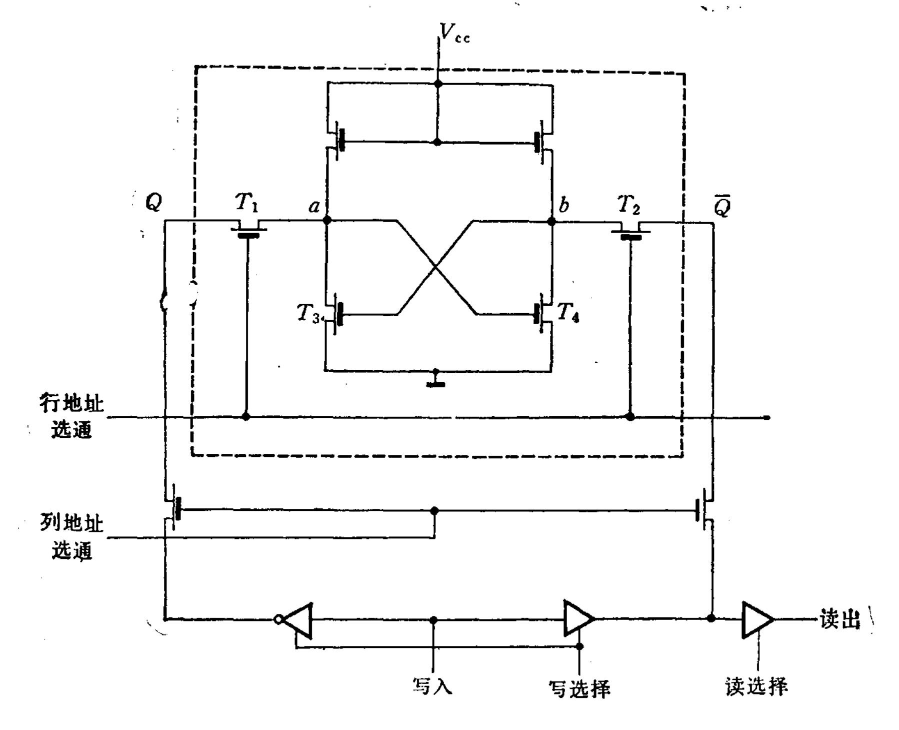


图 2-39 六管 SRAM

典型的6管CMOS存贮元件结构，由两个反相器构成。T_1，T_2是读写控制管，用译码器输出的高电平信号使它导通。若由Q端输入1，$\overline{Q}$端输入0，则T_4导通，b点为低电平，T_3截止，a点为高电平。在输入撤消后，由于反馈的作用，仍可稳定地保持上述状态。任何时候打开控制管，Q端均输出1，$\overline{Q}$端均输出0。存入0则情况相反。只要不断电或改写，存贮内容可以一直保持，所以称为"静态RAM"。

SRAM存贮数据可靠，使用简便，但因单元结构复杂，占面积大，芯片容量较低，耗电量大，速度也比较慢，6116，6264都是这种类型。

（2）动态RAM(DRAM)。为了提高集成度，增大内存容量，产生了一种更为简单的动态存贮单元电路，如图2-40所示。它是一个NMOS管的变型，除连接字线W的多晶硅栅极外，仅有一个N区通过电极连接位线B。另一个N区并不像通常那样以源极接电源，其表面绝缘层未开口，上面是一个类似栅极的硅电极C，并向外侧P区之上延伸。C极加一定的正电位，便能在N区旁形成一片毗连的反型层。这个反型层的表面同C电极间、底面同衬底间形成并联的两个电容，可用来存贮电荷，以电荷的有无表示数据1和0。当字线W有效，管子开启，便可以通过位线向这个电容存取数据。W无效时管子关断，存贮的电荷得以保存。这种简单的结构不能直接向电路送出足够幅度的数字电平信号，因而需要通过公用的读写电路存取数据。关于读写方式，留待第四章结合具体器件介绍。

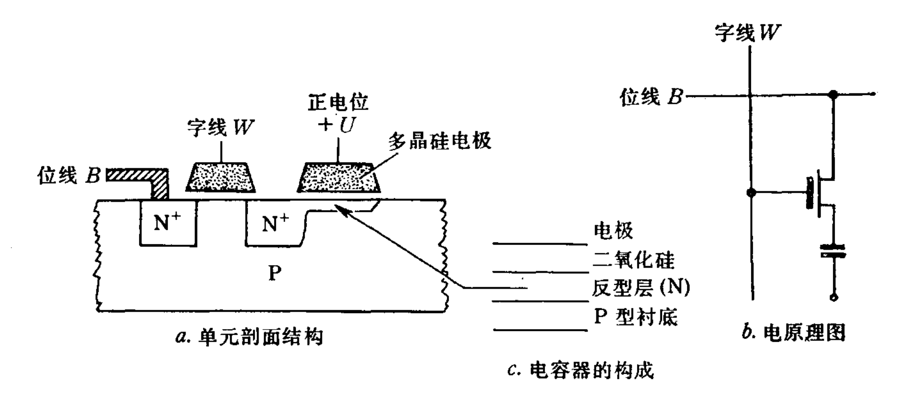


图2-40 单管DRAM

由于C电极的面积有限，形成的电容容量很小（约0.01ppF），电荷会以PN结漏电流的形式很快流失，数据保存只可能有毫秒级的瞬间，必须不断地对贮有电荷的电容器充电"刷新"，以动态地维持所存数据。DRAM具有高集成度、高速度和低功耗等优点，不过要以管理上的复杂性作为代价。4116、4164是常用的DRAM芯片。

各种存贮器中存贮单元的组织形式，分别采取"位结构"或"字结构"。位结构存贮器只有一位数据线，以每个存贮元件为基本存取单元，各有单独的地址，每次存取一位(bit)数据。字结构存贮器的数据线条数与字长一致，并行输入和输出数据，每字各位的存贮元件共用一个地址，不能"单独行动"，存取单位是字。常用的字结构存贮器存取单位为8bit——称一个"字节"(BYTE，缩写"B")。

对于CPU字长8位或16位的微型机，字节是信息交换的基本单位，因此习惯上以字节

为存贮容量的单位。容量很大时，还用到 KB(2^{10}B)和 MB(2^{20}B)等单位。从软件的角度，常把存放一字节数据的存贮空间(实际有 8 个单元电路)称为一个存贮单元。用位结构存贮器存放并行的字节信息，需要八片并列，各片同一位置的存贮单元用同一地址信号选通。这样的八个分散的 bit，使用时仍可视为一个单元。

存贮器 IC 成品的标称容量常以 Kbit 为单位。例如，6116 容量是 16K bit，也就是 2KB 27128容量 128K bit，即 16KB。

第三章　微处理器探秘

电子计算机的基本结构，可以分为控制器、运算器、存贮器和输入输出设备四部分。其中控制器和运算器合称中央处理单元(CPU)。它是计算机的中枢，电脑的功能最主要就是CPU的功能。现代集成电路技术，已经能够把整个CPU制作在一片(或几片)大规模集成电路芯片上。这就是微处理器(MPU)。微处理器和由它发展而成的单片微型计算机，正日益广泛地运用于生产、生活各领域，逐渐进入每一个家庭。

微处理器是逻辑电路的集大成者。MPU的内部结构非常复杂，但原理并不深奥，归根结底是由我们已经熟悉的逻辑部件组合而成。它的组成方法和工作机制，具有中学文化水平和逻辑电路基本知识的人都能理解。对它进行剖析，我们会看到一幅绚丽多彩的逻辑图画，从中学会用简单的逻辑元件组织起来实现复杂功能的技巧。了解了MPU，才算是真正懂得了计算机。

为了认识MPU，不妨重温一下它的历史。开辟计算机新纪元的微处理器，并非产生于任何理论、计划或预测，而是由一个看似偶然的商业事件所促成。1971年，美国英特尔公司曾为日本商社研制过一个计算器用的单片处理器INTEL 4004，并未引起多少注意。1972年，这个公司又受托研制一种用于显示器控制的8位微处理器芯片，命名8008。但因为它的工作速度低，竞争不过双极型产品，未被采用。当时研制者也没有意识到它还会有多大的价值。谁知投入市场后，由于它具有可贵的通用性，能用于各种控制系统，竟意外地畅销。英特尔公司敏锐地抓住契机，重建已经下马的研制班子，在一年之后就推出了成为现代微处理器里程牌的INTEL 8080。一时，各半导体厂家群起效尤，微处理器园地百花竞放，但设计上不同程度地都模仿了8080，有些简直就是8080"在电子扫描显微镜下的复制品"。

1976年，由8080主要设计人员另立门户的泽洛格公司，在8080基础上作了重大改进，推出了新一代8位微处理器Z80。它既兼容了8080的全部指令，性能又有很大的提高，可以说使八位MPU技术达到了极点。从那以后，微处理器开始步入十六位机时代，从8086发展到今天的32位高速微处理器80486。

从以上的历史可以看出，各种微处理器是一脉相承的。很多MPU具有相似的逻辑结构和工作方式，可以举一反三，容易触类旁通。虽然就世界微型机的发展来看，8位机已处于夕阳阶段，但目前国内拥有量较大的普及机、学习机、袖珍机、游戏机和单板机，仍以8位MPU(主要是Z80和6502)为主流，同一般电脑爱好者距离比较近。因此，了解微处理器仍应自8位机开始。

一、总体结构

(一) 外部引脚及功能

微处理器都是多脚片状封装的器件。引脚除供电气联结外，主要是作为同外部器件交换

信息的通道。除电源、时钟外，MPU 的引脚可分为"数据"、"地址"和"控制"三类。按照微型计算机的总线结构方式，它们都是用来连接印板上的对应总线的，引脚内是总线驱动接口电路。引脚实际已是外部总线的一部分，所以都以"总线"相称。

在计算机系统中，总线并非由 CPU 独占。存贮器和 I/O 接口要向总线发送数据。若系统是多处理器的，协处理器输出的数据、地址和控制信号也可驱动同一组总线。下面要讲到的"DMA 方式"也要"窃取"总线交换信息。在外部器件驱动总线时，CPU 的相应输出端口都应呈高阻态，因此它们的输出都应是三态的。

下面以 Z80 为主加以说明，引脚分配见图 3-1。

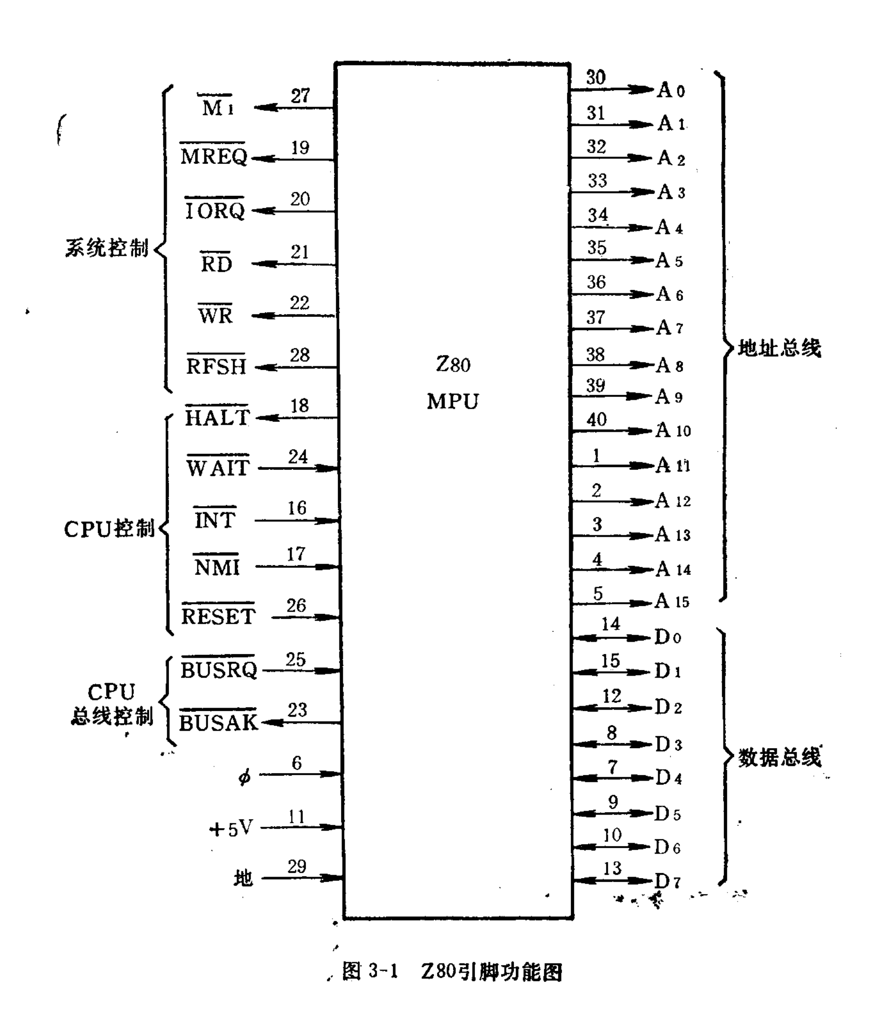


图 3-1 Z80引脚功能图

1. 电源和地线。

这是任何 IC 所必需的。8080 设计时，根据当时 INTEL 公司的 DRAM 技术要求，设置了 +5V、-5V 和 +12V 三种电源输入端。以后的 Z80、6502、8086 等都明智地改用 SRAM结构的内部寄存器，减为单电源（+5V）供电，只使用一条电源线。

地线连接电源负极，也就是测量电平的公共参考点。

2. 时钟线。

如果说电源是 MPU 作为电子器件的原动力，时钟就是它作为逻辑器件的原动力。时钟脉冲是使 MPU 按既定逻辑工作所必需的激励、定时和控制信号源。通常用来产生时钟脉冲的石英晶体振荡器，无法集成在半导体硅片上，需设在 MPU 外部。8080 和 6502 等 MPU 需要两相时钟，占用 2～3 个引脚。Z80 只需单相的 TTL 电平时钟信号 ϕ，用一个引脚输入。

3. 数据总线。

MPU 的工作有赖于从存贮器取得的指令代码。在执行指令时，往往还需要从存贮器或外部设备取得运算所需的数据。运算的结果一般也要送往输出设备或存入存贮器备以后使用。这些数据的交换通道是数据总线。为了高速工作，数据采取并行传送方式，数据线的宽度一般和 MPU 字长一致。Z80 的数据总线是 D7～D0，采用正逻辑信号，三态输入/输出。

4. 地址总线。

存贮数据的存贮器是一个庞大的单元阵列，外部设备也可有多个，因而必须由 MPU 发出特定的地址信号，才能指定所需的单元或设备。发出地址信号的是地址总线。地址总线的数量由 MPU 设计的寻址范围所决定。Z80 直接寻址范围为 64KB，因而地址总线A15～A0共 16 条，采用正逻辑信号，三态输出，用它们的不同组合表示 0000H～FFFFH 间的任一地址。

5. 控制总线。

为了将 MPU 和外部器件组成有机的系统，协调工作，它们之间需要交换一些非数量性的信息——控制信号。MPU 通过输出的一些控制信号控制外部器件的工作，使用者也可以通过一些输入控制信号控制 MPU 的工作。可使用的控制线的多少，一定程度上反映了 MPU 对系统的控制能力、应变能力和它的可操作性。小型计算机 CPU 的控制线多达数十条，MPU 受结构和引脚数限制很难与之相比。

Z80 的控制总线共 13 条。控制信号都采用负逻辑，低电平为有效，在信号名称上加横线表示（负逻辑信号名和"非"运算表达式形式相同，请注意区分）。

(1)CPU 控制。主要为输入信号，用来接收由外部电路发来的对 MPU 的控制信息，以作出响应。

① $\overline{\text{RESET}}$ 输入，复位信号。这个信号可使MPU进入初始状态并立即从头开始工作（若在工作过程中收到这一信号，将无条件地返回初始状态），即：PC = 0000H，中断允许触发器 $IFF_1 = IFF_2 = 0$，I = R = 00H，地址总线、数据总线端口均呈高阻态，$\overline{\text{BUSAK}}$有效，其余控制总线均无效。

② $\overline{\text{INT}}$ 输入，可屏蔽中断请求信号。在 MPU 正常工作进行中接收到这一信号，如果处于允许中断状态（内部的中断允许触发器 IFF = 1），则在当前指令执行完毕后作出响应——将 PC 当前内容（下一条指令地址）压入堆栈保存，并转入指定地址执行预先准备的中断服务程序，对引起中断的外部事件进行处理。若 IFF = 0 将使 $\overline{\text{INT}}$ 受到屏蔽，不予响应。

③ $\overline{\text{NMI}}$ 输入，不可屏蔽中断请求信号，以脉冲下降沿触发有效。当 $\overline{\text{NMI}}$ 信号线电平由高变低时，（如果 $\overline{\text{BUSRQ}}$ 无效）MPU 就会响应而不受中断屏蔽的影响。因而可用来处理外部紧急情况。$\overline{\text{NMI}}$ 中断方式固定以 0066H 地址为中断服务程序入口。

④ $\overline{\text{WAIT}}$ 输入，"请等待"信号，用于低速存贮器或外部设备同高速的 MPU 同步配合。设备不能按 MPU 的时序发送或接收数据时，可使 $\overline{\text{WAIT}}$ 电平变低，MPU 检测到以后便会

自动插入等待周期,直到对方准备就绪并使$\overline{\text{WAIT}}$信号无效,再进行读写操作。

⑤$\overline{\text{HALT}}$ 输出,暂停信号,表示 MPU 处于暂停状态。暂停状态期间不执行任何指令,因而不能以指令中止暂停。只有在响应中断和 DMA 请求时,暂停才能结束。

(2)系统控制。均为输出信号,MPU 通过它们对内存和外围设备进行控制。

①$\overline{\text{M1}}$ 输出,取指信号,表示MPU 进入取指令操作码周期。它同 $\overline{\text{IORQ}}$ 同时有效即是"中断应答信号",向外设表示响应其中断请求。

②$\overline{\text{MREQ}}$ 三态输出,存贮器请求信号,与十六位地址信号配合,选通所需的存贮器。

③$\overline{\text{IORQ}}$ 三态输出,I/O 请求信号,与低八位地址信号配合选通所需的I/O 口。

④$\overline{\text{RD}}$ 三态输出,读信号,用来控制上述选通的内存和 I/O 口输出数据。

⑤$\overline{\text{WR}}$ 三态输出,写信号,用来制控上述选通的内存和 I/O 口接收数据。

⑥$\overline{\text{RFSH}}$ 输出,刷新信号,与 $\overline{\text{MREQ}}$ 和 A6～A0七条地址线配合,在取指周期后半部分时间内对 DRAM 进行刷新操作。

(3)总线控制。一般情况下计算机内外的信息交换都以 CPU 为中介进行。为了便于大量数据的快速交换,Z80提供"直接存贮器存取(DMA)"方式,可使 MPU 与总线暂时脱离,由内存同外设在DMA控制器的控制下直接交换数据。

①$\overline{\text{BUSRQ}}$ 输入,总线请求信号,可使MPU 的地址总线、数据总线和输出控制线均呈高阻无效状态,以便外设和内存"窃取"总线直接交换信息。

②$\overline{\text{BUSAK}}$ 输出,表示MPU已响应$\overline{\text{BUSRQ}}$,总线可供DMA占用。

试比较一下常见微型计算机的 MPU,无论8080,Z80,6502,还是十六位的 8086,外观竟基本一样,都是封装在同样的40脚双列直插式管壳之中。40脚并非是MPU的最佳选择,只是因为在那个年代,价值昂贵的IC成品测试仪只能测试40～42个管脚的器件,采用新的仪器将使芯片成本大为上升。但40只引脚对于功能完备的微处理器来说是非常局促的。就 8 位MPU而言,必不可少的电源、时钟、数据和地址线便需要27～30条,只有10条左右供控制信号使用。这就迫使设计者们不得不采取"削足适履"的种种对策。对引脚的不同使用方式,也是构成各种 MPU 的差别之一。随着IC生产的发展,70年代末以后,产品测试条件大为改善,40脚的限制已经突破。80286就使用了 68 个引脚,增加的地址线使得它的直接寻址范围达到16 MB 之多。

(二) 内部结构

常用的几种MPU提供的结构框图虽然看起来差别很大,但基本逻辑却是相似的,主要由控制器、运算器、寄存器三大部分和输入输出接口电路组成,它们都围绕着一组 8 位并行的内部数据总线。图 3－2 是Z80逻辑图。

1. 数据通道。

运算器、数据寄存器和内部数据总线组成 MPU 的数据通道。运算器是主要的执行部件,计算机的数据处理主要由运算器完成。数据寄存器用来暂时存放待操作的数据或中间结果。数据通道的各个节点有若干可控的逻辑门电路。控制器通过它们左右数据的选择和流向,从而实现所需的运算、传送操作。

8 位MPU的内部,较普遍的是只用一组数据总线,即单总线结构。单总线的优点是占用

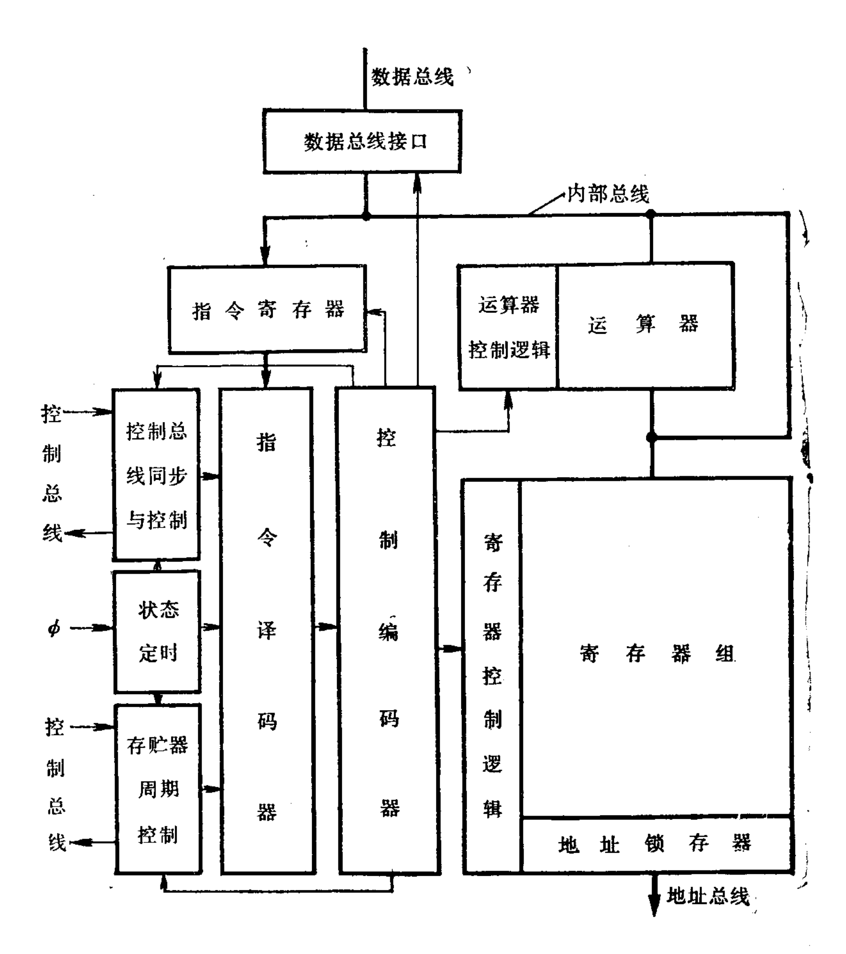


图 3-2　Z80逻辑图

芯片面积小，但这是以较慢的工作速度作为代价的。一次运算的两个操作数送入运算器，运算器送出结果，必须分三次使用同一组总线才能完成。双总线MPU增加了一组"结果总线"，三总线MPU更进一步将两个操作数各用一组总线传送，因而可以提高工作速度。

运算器、寄存器组和数据总线缓冲器都需要向内部总线输出数据，它们的输出端可以看作是以"线或"方式连接在一起的。根据工作条件，可采用三态型或集电极开路型输出驱动器同总线连接。

2．地址通道。

地址寄存器和内部地址总线组成MPU的地址通道，向外部地址总线发送地址信号。地址总线的宽度一般是16位。

地址寄存器有的是专门的，也有的由数据寄存器兼任，称为"通用寄存器"。通用寄存器可以看作数据通道和地址通道间的接口，数据能够通过它转化为地址。

3．控制逻辑。

控制器和控制逻辑电路是MPU的神经系统，从面积看也占据着芯片的主体部分。控制器接受输入的指令及外部控制信号，译码产生成组的内部控制信号，经控制线送往数据通道和地

址通道的各个门电路，控制它们的工作。

4. 输入/输出接口

MPU的数据总线一般都是输入输出共用，以减少引脚和布线。为了使输出数据能在总线上保持足够的时间，需用数据总线锁存器(DBR)将输出数据锁存，以待外部器件完成写入操作。输入数据进入内部总线后，即可直接装入数据寄存器。数据总线接口电路同时面向内外总线，双向传送数据，多采用双向三态驱动器结构。

MPU的地址信号只是向外输出，总线接口采用单向三态驱动器。Z80等MPU的地址信号须在数据存取全过程中稳定地保持在地址总线上，所以用地址锁存器(ADR)锁存，输出缓冲器通常都处于允许输出状态。来自寄存器组的16位地址数据存入锁存器后，立即就会出现在地址总线上，并一直保持到新的地址进入。因而地址总线上总有一组地址信号。只在MPU响应外部设备的总线请求信号后，才会使地址缓冲器输出呈高阻态。

MPU的总线缓冲器的输出电流具有一定的驱动负载能力。如Z80每个输出端为低电平时输出电流1.8mA，动态驱动能力为130ppF。

常用的低功耗肖基特TTL电路(74LS系列IC)，输入电容很小，主要是静态的电阻性负载。每个输入端在低电平时的输入电流约0.4mA。所以Z80只能直接驱动4个这样的负载，若配接更多个，则应加接驱动器，否则会造成数字电平幅度不足。常用的CMOS器件(如存贮器等)，输入电流仅1μA，从静态来看驱动上百个也不成问题。但CMOS输入电容约5ppF，主要是动态负载，Z80只能驱动20余个(这还只考虑了逻辑电路本身的电容 外部线路的分布杂散电容尚未计算在内)，否则在输出数据变化时因充放电时间延长而不能保证正常的工作时序。

二、运算器和运算方法

(一) 运算器结构

运算器逻辑框图见图3-3。其主体是算术逻辑单元(ALU)。

MPU的运算功能较多。以Z80为例，它能进行以下各种基本运算：加、减、与、或、异或、比较、位测试、位置位、位复位、递增、递减、左右移位和循环。为完成这些运算而各设一组专用的电路是不明智的。运算器设计的基本原则便是一“器”多用，以一个综合性的电路完成多种运算。

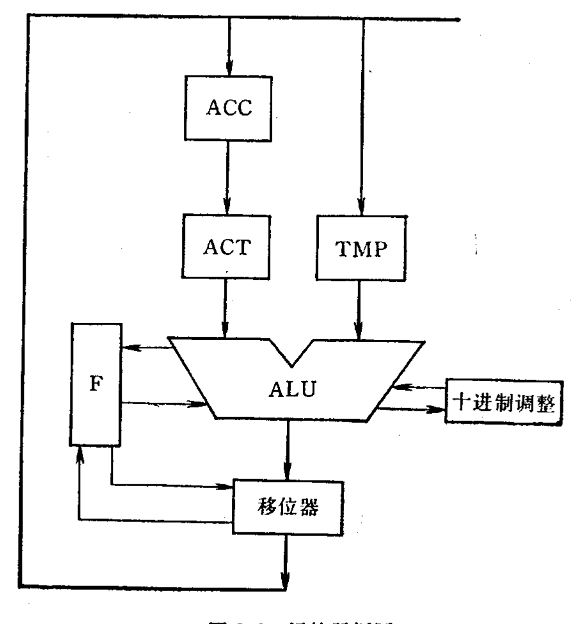


图3-3 运算器框图

各种MPU都有至少一个“累加器(ACC)”。它主要用来存放一个操作数并在运算后存放结果。更确切地说，应称为“累加寄存器”。有的MPU累加器也可担负部分运算工作，如取反、移位等。累加器通常挂在内部总线上，并且能同

一般数据寄存器同样地编址使用，这时它可视作寄存器组中的一个通用寄存器。在具有多个通用寄存器的MPU中，累加器的特点在于它可以通过多路分配器向 ALU 的一端直接送入数据。这种结构有助于减少单总线结构造成的时间损失——当累加器向ALU送操作数时，另一操作数便可以同时通过总线送入。

ALU有两组输入端(我们将接受累加器输入的称为*A*端，另一端称为*B*端)。将它们都直接连在总线上将无法同时获得两个操作数。而且ALU是组合逻辑电路，不按时钟同步工作，输入端加上信号后随即有结果送上总线——对于单总线的MPU，也就是又反馈到ALU输入端(称为“临界追赶”)，导致产生新的伪“结果”。所以需要在总线和ALU输入端之间设置时序缓冲部件。*B*端设一数据暂存器TMP。*A*端可直接以累加器作缓冲寄存器，但这样就形成累加器“独霸”*A*端的局面，使得不涉及累加器的运算(如通用寄存器内容的相加)成为不可能。改进的办法是在*A*端设一累加器锁存器(ACT)。它能从累加器和数据总线两方面接受数据，并实现总线缓冲。

ALU 是一个综合性的算术逻辑运算电路。它的输出端接有移位器(SHIFT)，可将其输出数据进行移位处理，然后送入内部总线。ALU和移位器的输出的某些状态信息，经专门的电路送入状态标志寄存器F存贮，以便参加后续的运算操作或供指令测试。此外还有十进制调整电路(DAA)。

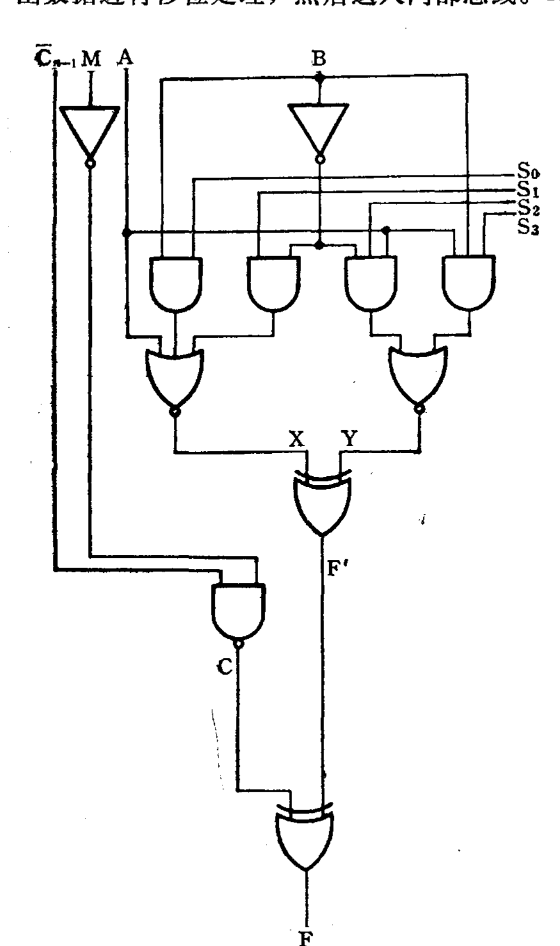


图 3-5 ALU 的一位运算电路

(二) ALU

各种MPU的运算功能不同，ALU的结构也有差别。下面，我们以一个通用的 ALU 电路为例，来说明运算原理。

ALU 实际上是在全加器基础上的扩展。上一章讨论的全加器可作加法运算，增加求反电路后也只能作减法。综合性算术逻辑运算部件的基本方案，是将全加器的“变量输入”逻辑变为“函数输入”逻辑，通过可选项的组合函数实现各种基本逻辑和算术运算。在本节的讨论中，为了区别算术加和逻辑加，它们的运算符号分别用“+”和“∨”表示。“·”(或省略)仍表示逻辑乘，“—”表示算术减。

图 3-4 (下页)是一个四位通用型 ALU 的逻辑图。它可分别用于正、负逻辑系统，下面我们只讨论输入输出都是正逻辑的情况。电路由输入逻辑、全加器和进位逻辑三部分组成。

1. 输入逻辑和全加器。

图 3-5 是这个ALU中一位运算电路的略图。前端是两个多路选择器构成的输入函数发生器。为了简化结构，电路中采用与或非门，其



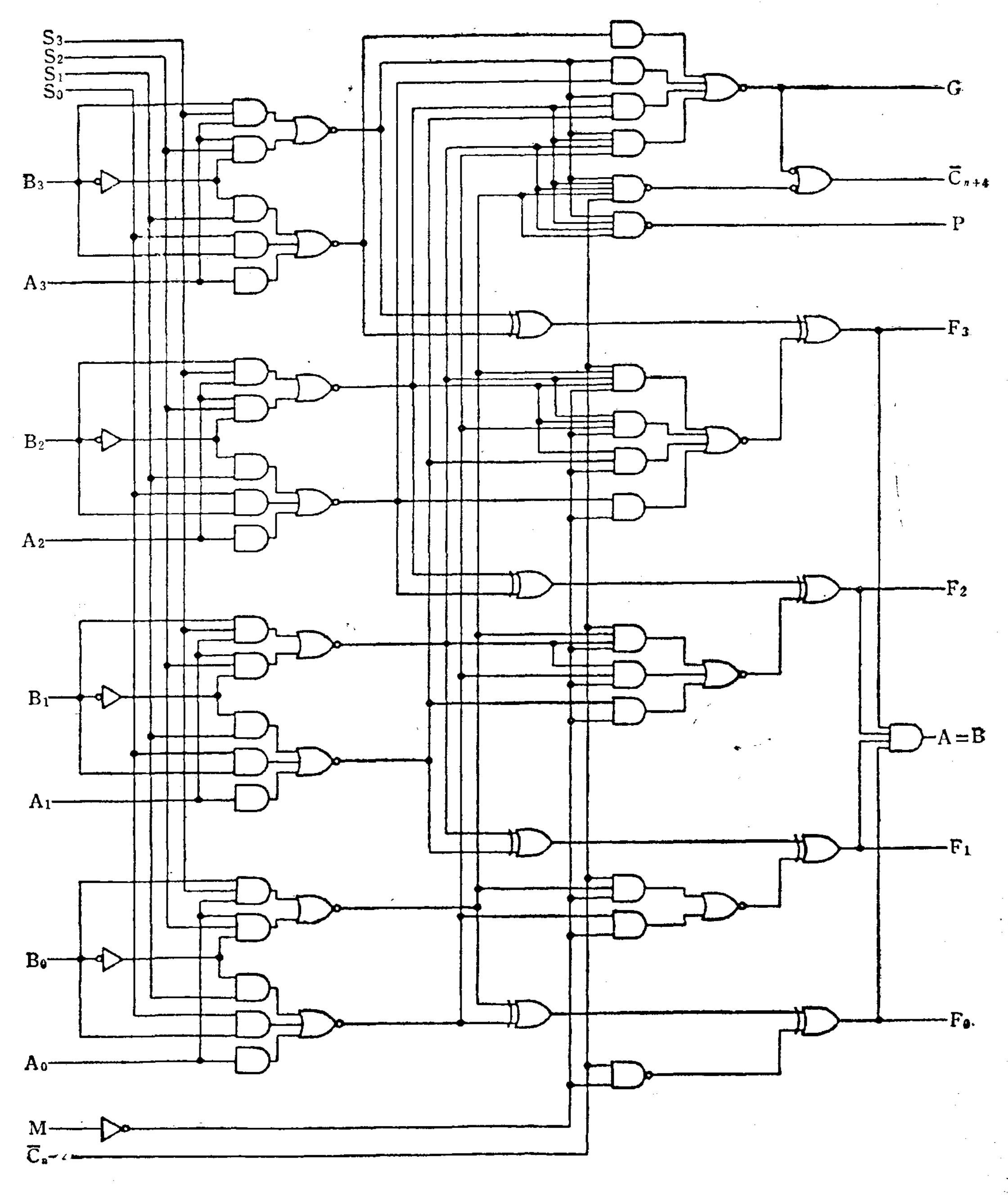


图 3-4　四位 ALU 逻辑图

输出是与或函数的反值，函数式分别是

$$X = \overline{A \vee S0B \vee S1\overline{B}}$$

$$Y = \overline{S2A\overline{B} \vee S3AB}$$

选择信号S(S3～S0的不同组合)可任选其中一个或几个变量参加运算，从而在第一个异

或门输出端获得不同的函数值$F' = X\oplus Y$。因为异或门对正负逻辑等效，所以$\overline{X}\oplus\overline{Y} = X\oplus Y$。

例如，当S = 1001时，$X = \overline{A\vee B}, Y = \overline{AB}$。所以

$F' = (A\vee B)\oplus AB = A\overline{A}\vee A\overline{B}\vee\overline{A}B\vee\overline{B}B = A\overline{B}\vee\overline{A}B = A\oplus B$。

这时，从输入端到第一异或门的组合逻辑相当于对于A和B的半加器，F′是A和B的半加和。

进位信号C由低位来的进位$\overline{C}_{n-1}$和算术/逻辑选择信号M经与非门产生。进行算术运算时令M = 0，反相后使C完全由$\overline{C}_{n-1}$决定。$\overline{C}_{n-1} = 1$表示低位无进位，C = 0。$F = F'\oplus 0 = F' = A\oplus B$，即本位的半加和就是A、B的全加和。$\overline{C}_{n-1} = 0$（有进位）时C = 1，$F = F'\oplus 1 = \overline{F'} = \overline{A\oplus B}$，本位全加和等于半加和的反值，实现了按位全加运算（0 + 1 = 1，1 + 1 = 0，这里未考虑向高位的进位问题）。

又如，当S = 0110时，$X = \overline{A\vee\overline{B}}, Y = \overline{A\overline{B}}, F' = (A\vee\overline{B})\oplus A\overline{B}$。

它和A + B不同处仅在于$\overline{B}$取代B，相当于加B的反码。同时令$\overline{C}_{n-1} = 0$（进1），便可实现"补码加"形式的减法运算。

那么逻辑运算怎样进行呢？例如，当S = 1011时，有

$F' = (A\vee B\vee\overline{B})\oplus AB = (A\vee 1)\oplus AB = 1\oplus AB = \overline{AB}$。

这时F′可看作目标函数（这里是与函数）的反值。因为逻辑运算是对位运算，不进位，令M = 1，反相后就封锁了$\overline{C}_{n-1}$的作用，并使与非门输出为0，第二异或门便相当于反相器，$F = \overline{F'} = AB$。

在M = 1的条件下，用S = 1011就实现了逻辑"与"运算。同理，用S = 1110可实现"或"运算，S = 0110可实现"异或"运算，等等。

2．运算选择。

对于上述结构的ALU，进行的运算种类由运算选择信号S3S2S1S0、算术/逻辑选择信号M和低端进位信号$\overline{C}_{n-1}$共同决定的，可有$4^2 + 4^2\times 2 = 48$种变化。每种对应于一个逻辑函数或算术运算。不过对于通用运算器来说，其中很多"运算"是冗余的，并无实用价值。可供使用的运算种类见表3－1。

实际使用的ALU，根据对运算种类的要求，结构还可以比通用型简化。

3．进位逻辑。

按前章全加器的基本逻辑，各位的进位是从低到高依次发生、异步传递的。FA_n和FA_{n-1}同时进行运算，FA_n最初的结果$F_n = A_n + B_n$。若FA_{n-1}产生进位，FA_n相当于又要"重新"运算一次，才能产生完整的全加和。之后，FA_n的进位又才送到FA_{n+1}参加运算……当全加器位数较多时，这种串行方式的进位传递过程将造成较大的延时，使运算速度降低。解决这个问题的方法之一是采取"先行进位"方式。

所谓"先行进位"，就是将"求本位和"与"求进位值"同时分别进行。上面讨论的求本位和的电路，不包含进位成分。由于二进制加法进位规律简单，进位不难用进位函数及相应的组合逻辑电路直接求出。

全加器第n位向第$n+1$位的进位C_n，可用以下函数表示：

$C_n = G_n\vee P_nC_{n-1}$。

表 3-1

选择 $S_3\ S_2\ S_1\ S_0$	逻辑运算 M = 1	算术运算 (M = 0) $\overline{C}_{n-1} = 1$ (低端无进位)	$\overline{C}_{n-1} = 0$ (低端有进位)
0 0 0 0	$F = \overline{A}$		$F = A + 1$
0 0 0 1			$F = (A \vee B) + 1$
0 1 1 0	$F = A \oplus B$	$F = A - B - 1$	$F = A - B$
1 0 0 1		$F = A + B$	$F = A + B + 1$
1 0 1 0	$F = B$		
1 0 1 1	$F = A \cdot B$	$F = (A \cdot B) - 1$	
1 1 1 0	$F = A \vee B$		
1 1 1 1	$F = A$	$F = A - 1$	

G_n是"进位产生函数",$G_n = A_nB_n$,当两个操作数都为1时,必然向高位进位。P_n是"进位传递函数",$P_n = A_n \vee B_n$,当P_n为1时即使本位并不产生进位, 也会将低位来的C_{n-1}"传递"到高位去。若采取负逻辑,$G_n = \overline{A_nB_n}$,$P_n = \overline{A_n \vee B_n}$,可见它们就是加减运算时的两个输入函数,$G = Y$,$P = X$。因此进位运算可以和第一次半加平行地进行。

同理,第$n+1$位的进位函数为:

$C_{n+1} = G_{n+1} \vee P_{n+1}C_n$。

将上式代入,则有

$$C_{n+1} = G_{n+1} \vee P_{n+1}(G_n \vee P_nC_{n-1})$$
$$= G_{n+1} \vee P_{n+1}G_n \vee P_{n+1}P_nC_{n-1}$$

…………

逐级代入,每一位的进位函数都可由本位和低侧各位的输入函数以及最低进位 C_{n-1}同时求出,与全加运算同步进行。

图3-4包含了四位的"0级先行进位"逻辑电路,输入为负逻辑。最低进位$\overline{C}_{n-1}$和算术/逻辑选择信号M参加每一位的进位函数生成。各位进位函数值由与或非门输出后变为正逻辑信号参加第二半加运算。

显然,这种进位电路中位次越高,所需的与门越多。当ALU位数较多时,结构将变得相当庞杂。所以一般采取分级进位方式。如8位ALU由如图的两个内部先行进位的4位ALU组成,它们之间实行串行进位。16位ALU可由四个4位ALU组成,再用一组先行进位电路处理级间的进位关系。这样可以使速度和结构指标兼顾。因此,上述四位ALU常作为一种基本组件。它除输出本组各位的运算结果外,还输出本组的C,P,G函数供级间进位电路使用。

Z80的一个出人意料之处是 ALU 只有四位。这显然是由于规模较大的寄存器组和控制存贮器占用了芯片更多面积所致。这样,一个8位数据的运算就必须分两步进行。每次运算时,ACT和TMP通过二选一多路选择器先向ALU送入操作数的低4位。半字节运算的结果暂存于输出缓冲寄存器,产生的进位存入"半进位标志"触发器H。然后送入两个操作数的高

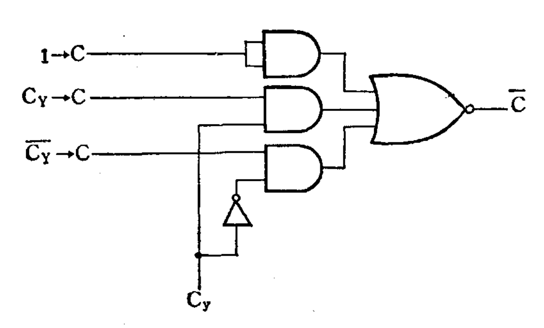


图 3-6 ALU 低端进位控制

4 位，半进位值也参加运算，结果与寄存的低 4 位合并后输出，于是总线上出现完整的运算结果。由于运算器工作速度快，用时钟脉冲正负半周各控制送入半字节数据，能在一个时钟周期内完成两步运算。因而使用者仍可将它看作八位ALU。

（三） 运算功能的扩充

当要求 MPU 具备更多运算功能时，上述 ALU 还不能完全满足需要。增加一些辅助逻辑电路，可以使运算功能得到扩充。

1. 低端进位控制。

ALU最低端的进位输入$\overline{C}$，是一个重要的数据。用如图 3-6 所示的电路产生$\overline{C}$的输入，配合适当的运算选择码就能构成不同的运算类型。

（1）$C_y \to \overline{C}$：送入C标志触发器内容，以实现带C标志加法（$A+B+C_y$）运算。

（2）$\overline{C}_y \to \overline{C}$：送入C标志的反码，以实现带C标志减法（$A-B-C_y$）运算。

（3）$0 \to \overline{C}$：送入0，以实现A + 1 和不带C标志减法（A − B）运算。

（4）以上各控制信号都无效：送入 1，以实现A − 1 和不带C标志加法（A + B）运算。

2. 送数控制。

在ALU的输入端，用一组控制输入的门电路选择送入运算数据、全0或全 1 。当A端各位送入全0时，用($A \vee B$) + 1 的运算便实现了B + 1 。A端送入全 1，用($A \cdot B$) − 1 的运算也就实现了B − 1 。此外它还可用于位测试。

3. 位寻址与位操作。

为了实现对一个操作数中指定位的置位、复位和测试，需要有一个位寻址电路，如图 3-7 所示，用 3 位“位地址码”译码产生位选择信号，再配合SET/$\overline{\text{RES}}$信号向目标位送入 1 或0，非选择位数码不变。置位/复位电路可设在 ALU 输入端，也可设在输出端。利用表 3-1 中的 F = A或F = B的运算，就可把单操作数传送通过全加器。

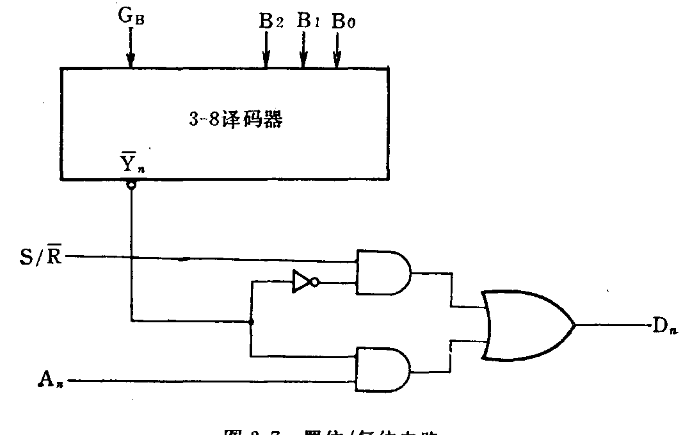


图 3-7 置位/复位电路

为节省器件，位测试可以不用专门的测试电路，而通过与运算来实现。一个输入端送入待测试数，另一输入端送入一个“测试掩码”。掩码的产生方法，是由送数电路送入全0并将待测试位置位为1。与运算后，非测试位都输出0，被测试位若为0则运算结果为0，被测试位为1则结果非0。结果状态只记录在状态标志寄存器的Z标志位。

4．十六位二进制数的运算。

有的8位MPU具有的16位算术运算指令，实际上是作为两个8位运算分两步完成的。例如Z80的ADD HL，DE指令，具体操作过程是：

(1)E→ACT。

(2)L→TMP，(ACT)+(TMP)→ALU。

(3)ALU→L、C_y。

(4)D→ACT。

(5)H→TMP，(ACT)+(TMP)+C_y→ALU。

(6)ALU→H、C_y。

16位寄存器对的加1、减1运算，在寄存器组设有“增量/减量器”的MPU(如Z80)，可不经过ALU，直接由增量/减量器完成。

5．十进制运算。

MPU大都具有对BCD码进行运算的能力。BCD码是一种用二进制数表示十进制数的代码，用4位二进制数表示十进制的一位数码，即0000B=0，0001B=1，……1001B=9。BCD码的运算先按二进制数进行，然后将结果调整为BCD码。为此运算器需附有BCD调整电路。

两个BCD数相加减后，如果有半字节＞1001(9)，或进位标志、半进位标志置位，都说明上一次运算发生了BCD溢出，需要对结果重新调整，恢复为正确的BCD数。调整的方法是再进行一次加或减操作。当$(D1 \vee D2) \cdot D3 \vee H = 1$或$(D5 \vee D6) \cdot D3 \vee C_y = 1$时，将根据减标志选择在相应半字节加上或减去一个调整码0110B，不需调整的加0000B，就变成正确的BCD数。8080和Z80进行BCD数运算，都用普通算术运算指令后跟一条BCD码调整指令来完成。8080只能对加运算结果进行调整。6502的加减指令是二进制和十进制运算兼用的，将十进制运算标志置位后，将自动启动调整电路，产生正确的BCD结果。

6．移位器。

移位是算术运算和数据处理的重要手段。如乘法运算需将被乘数左移，除法运算需将被除数右移。既需要将数据有序移动又不能丢失其中信息时，可采取首尾相续的循环移位。进位标志C_y在移位操作中常常被作为“第九个位”使用。它可以接受一端被移出位的内容，也可向另一端的位移入自己的内容，在循环移位中作为首、末位间的中介，构成九位“大循环”。

移位操作可在移位寄存器中进行，但装入、移位、送出都需按时钟同步操作，耗时较多。常用的组合逻辑移位器原理见图3-8。它是一个数据多路分配器，控制数据送入同位、高位或低

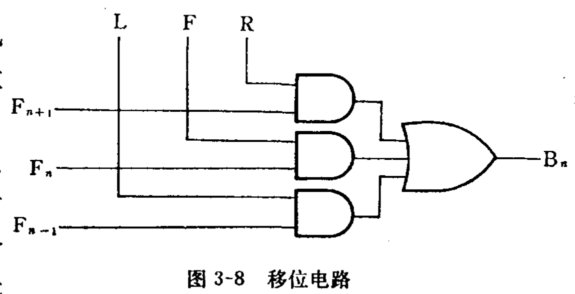


图3-8 移位电路

位的内部数据总线，从而达到移位的目的。用逻辑网络将 ALU 输出的最高位、最低位、总线以及进位标志触发器 C_y 联系起来，使数据能在它们之间按规定移动，就能实现各种移位和循环移位操作。

移位器设在全加器的输出端，是具有乘除和浮点运算功能的 CPU 所必需的，因为一次加减操作后常需立即移位以形成部分积等中间结果。对于移位器在 ALU 输出端的，ALU 便成为传送数据的通道，用表 3-1 中 F = A，F = B 的运算，可以将操作数送入移位器。

在 ALU 输出的F6～F1和总线的D7～D0之间，只有三种选择——直送($F_n \to D_n$)，左移一位($F_n \to D_{n+1}$)和右移一位($F_n \to D_{n-1}$)。各种移位和循环移位操作的区别，主要在于 F7 或 F0 的移出去向，D7 或 D0 的移入来源。除右移时 F7→D6、左移时 F0→D1 外，还可有

C_y←F7(所有左移和左循环)，D0←F7(不带 C 标志左循环)。

C_y→D7(所有右移和右循环)，D0→D7(不带 C 标志右循环)，F7→D7(算术右移)。

F0→C_y(所有右移和右循环)，F0→D7(不带 C 标志右循环)。

D0←C_y(带 C 标志左循环)，D0←F7(不带 C 标志左循环)，D0←0(算术左移)。

(四)状态标志寄存器

ALU 进行的运算，除产生一个数据结果，经总线送入寄存器或存贮器外，还会产生另一种结果——反映运算和数据结果特征的一组状态信息("状态字")。它们由 ALU 的输出端直接送入状态标志寄存器"记载"下来，供后续的操作参考和使用。例如全加器作算术运算时最高位的进位状态，数据总线和寄存器都不能容纳，但它相当于笔算时打的进位点或借位点，必须记录下来，否则多字节的加减运算便不能正确进行。这就是上面已经讲到的进位标志。有的运算并不产生数据结果，如比较和测试的结果就只是影响状态标志，我们只能从零标志和进位标志上去了解比较和测试的结果如何。状态标志的最主要用途，是作为判断程序是否需要转移的逻辑条件。

每个标志寄存器是一个独立的触发器。它们的输入端通过不同的逻辑网络接在 ALU 的输出端。各种运算操作的控制信号中包含着置有关标志位的信号，以选择某一状态的记存与否。各标志触发器的输出送给控制器的状态检测电路。

各种 MPU 状态标志寄存器所设的标志位及其符号不完全相同，有的还包括非运算状态(如中断标志、暂停标志等)。Z80 的状态标志寄存器 F 共 6 位，全部是运算标志，标志字结构见图 3-9。

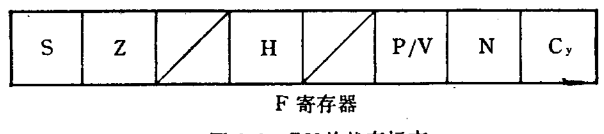


图 3-9 Z80的状态标志

1. 进位标志Cy。

它是标志寄存器中使用最频繁的一个。图 3-10 表示了它的输入逻辑关系。事实上，它常被当作一位数据寄存器使用。

2. 零标志Z。

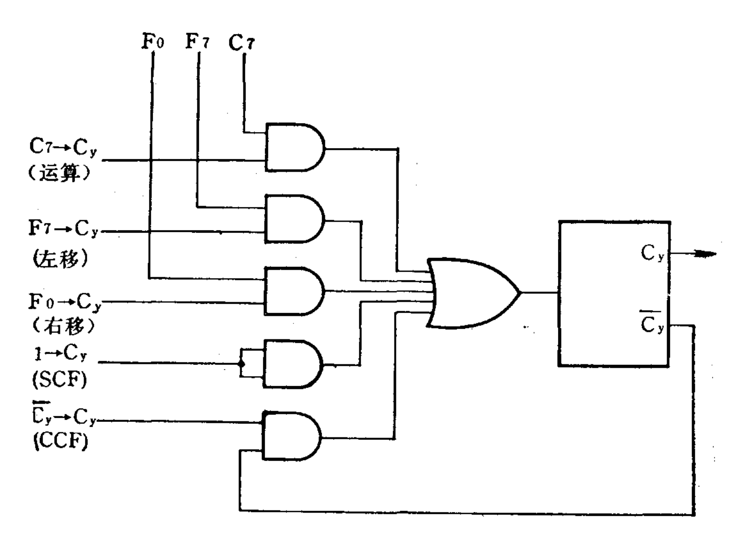


图 3-10 C标志触发器输入逻辑

它由ALU的8个数据输出端经或非门置位。一次运算结果8位全为0时Z=1(零状态为真),任一位或多位非零则Z=0。

3. 符号标志S。

它是ALU输出最高位(符号位)的副本。对于8位有符号数,运算结果为正(F7=0)时,符号标志为0;结果为负(F7=1)时,符号标志为1。所以它也称负标志。

4. 奇偶标志P。

它用来记录逻辑运算和移位操作后的二进制位中1的个数是否偶数,输出通过组合的异或门置标志值,1的个数为偶数置1,奇数置0。主要用于数据传输后检验是否有错。现在常用的计算机通信接口电路一般都具有奇偶效验功能,MPU此标志已不常使用。

5. 溢出标志V。

一个8位有符号数的绝对值只有7位,用来表示-128～+127范围的数。但因MPU对有符号数的加减运算和无符号数是同样处理的,其结果就可能由于FA6的不适当进位而超出这一范围,改变了符号位(称为溢出)而产生错误的结果。为便于监视这种情况和采取调整措施,MPU设置了溢出标志。它由全加器FA7和FA6的进位输出C7、C6经异或门置位。发生溢出置1,未溢出置0。

在Z80中,溢出标志V和奇偶标志P共用一个触发器,由算术/逻辑运算选择信号M控制置位(图3-11)。

6. 半进位标志H。

这是Z80特有的一个标志,用来记存4位ALU进行低4位运算时产生的进位值。这个标志可供BCD码调整电路使用,不提供用户以指令测试。

7. 减标志N。

这是Z80专为减法运算后对BCD数调整而设,由减运算控制信号置位。这个标志不提供用户以指令测试。

条件转移(包括跳转、转入子程序和返回)指令的执行,需要对标志状态进行测试。可测试

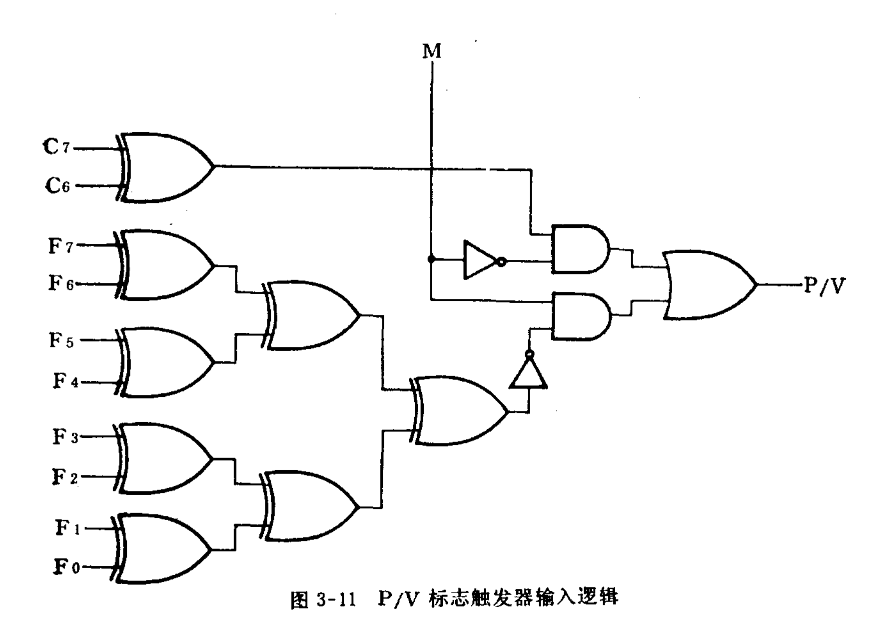


图 3-11 P/V 标志触发器输入逻辑

的标志用二位“标志码”指出。例如 Z80 转移类指令中：00—Z，01—C，10—P/V，11—S。另用一位逻辑量指出转移所依据的标志位状态，如 000—NZ，Z 标志 = 0 则转移；001—Z，Z标志 = 1 则转移。标志位的测试电路原理见图 3-12。位选择信号有效，指定标志位数据送入，符合逻辑条件时为 0，不符合逻辑条件为 1，它们可控制操作是否转移（见“控制器”一节）。

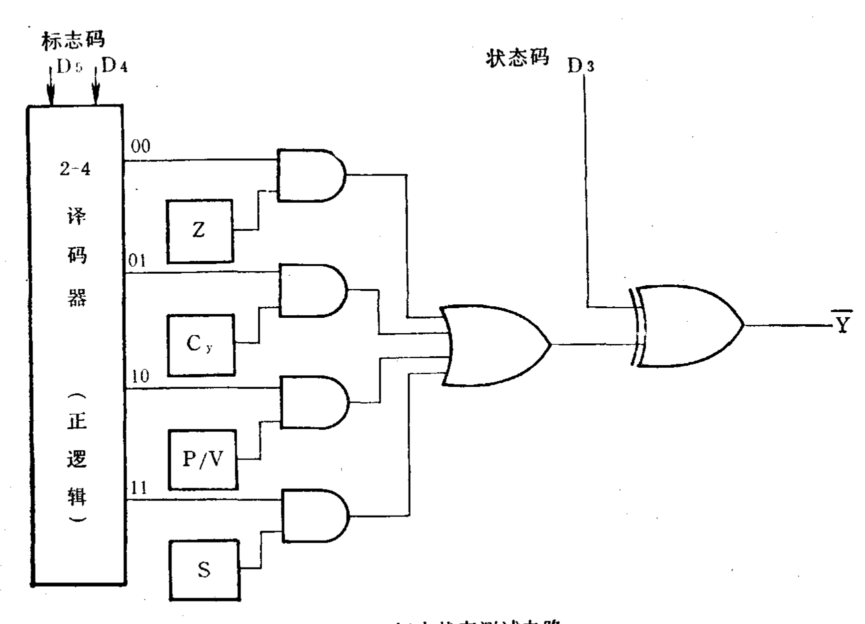


图 3-12 标志状态测试电路

三、寄存器

MPU 对数据进行某种运算或处理，常常不是一步操作所能完成的。操作的中间结果如果都用外部存贮器保存，存取很费时间，会降低运算速度。因此，MPU 均在内部设置若干快速寄存器，暂时存放中间数据，称为数据寄存器。数据寄存器可由用户以指令分配使用。MPU 访问内存时需要提供地址。由于指令和数据存贮方式的有序性，它们的地址具有以渐变为主的特征。将一个初始地址存放于专门的寄存器中，每次进行增量或减量便能获得下一个有效地址。当整个地址序列需要变动时，只需存入新的首地址即可。这种用途的寄存器称为地址寄存器。实际上，很多 MPU 的数据寄存器也能作为暂时性的地址寄存器使用，称为通用寄存器。通过它们可将数据直接变成地址，并能对地址进行计算处理。

寄存器的多寡是各种 MPU 的主要区别之一，但寄存器的种类却大体相近。常用的 8 位 MPU 中，6502 寄存器较少，以累加器兼作通用寄存器。它以灵活快速的存贮器操作作为弥补。Z80 是寄存器较多的一种 MPU。因为寄存器存取的指令短、执行快，多寄存器在进行较复杂的数据处理时无疑是一种优点(即使如此，程序员还常有感到 Z80 寄存器不够用的时候)。

(一) 寄存器组的结构

独立的寄存器通常采用第二章介绍的各种结构。多寄存器的 MPU 如 8080 和 Z80，为了简化结构，实际是在 MPU 内部制作一个 RAM 区，集中担负主要的寄存器功能。8080 的寄存器组采用 DRAM 结构，Z80 则采用 SRAM 结构。

1. 存贮矩阵。

MPU 内部 RAM 结构与一般的 RAM 器件相似，这里主要分析寄存器组的结构特点。图 3-13 是一种基本的结构形式。它的存贮矩阵分为若干行，每行 16 个存贮元件。一条行线的选择信号有效时，这一行各单元电路都被打开，可以通过列 I/O 线读写数据。这样的一维矩

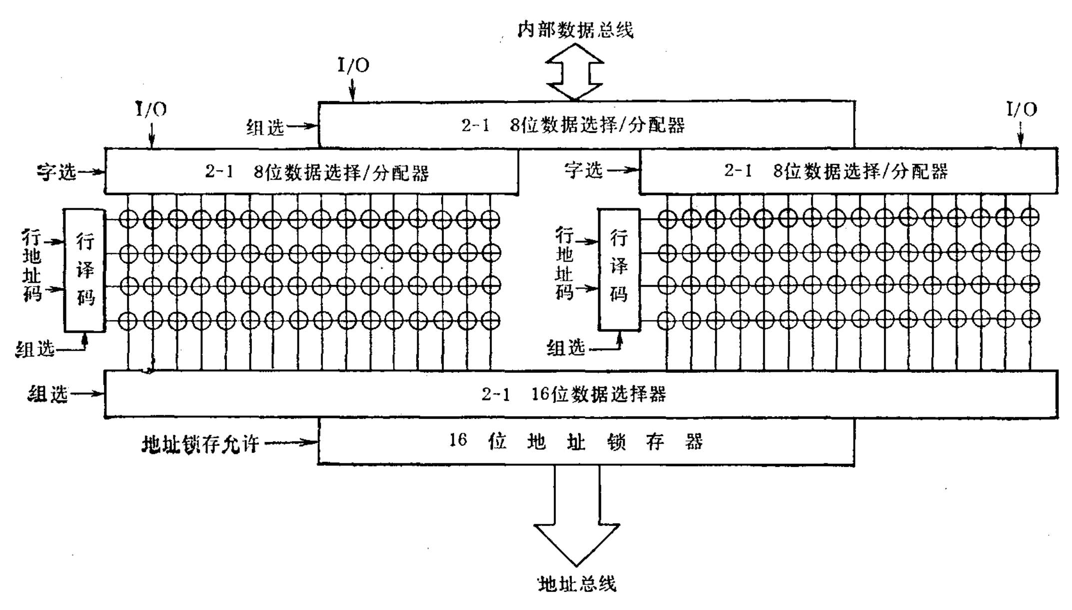


图 3-13 通用寄存器组结构

阵，每一行可以看作一个16位的寄存器，适合作地址寄存器使用。

8位MPU的数据寄存器只需8位，而且数据通道宽度也只容许8位。所以这个矩阵的一行可作为两个寄存器（或称寄存器对）使用。在行选择有效的条件下，输入/输出端口再用一组二选一8位数据选择/分配器，进行字（高8位或低8位）选择读写。

采取每行16位而不是8位的结构，主要是为了作为通用寄存器。I/O线的一端通过多路开关连接数据总线，另一端则作为地址信号通道。每一行既可作为一对数据寄存器，又可作为一个地址寄存器。于是产生了一种有趣的现象——从地址输出端看进去，内部RAM是16位地址寄存器，而从数据输入/输出端口看进去，又都是8位数据寄存器。正是这种结构，使寄存器组成了数据总线和地址总线间信息转换的接口。

虽然原则上这样的二维矩阵可以包纳的寄存器种类和数量都可以很多，但因为地址码的唯一性和I/O线的公用性限制，每次只允许使用一个寄存器。这就使得不同种类和用途的寄存器也不能同时使用，影响MPU的工作速度。因而综合性寄存器组常采取三维结构。寄存器分为若干组，每组是一个两维矩阵。各组的数据输入/输出通过多路开关接入总线。只要把可能同时使用的寄存器安排在不同的组内，就可避免上述问题。

2. 寻址方式。

上述三维存贮矩阵需用两种寻址部件重合寻址，用组、行地址译码器产生组、行选通信号，双重的多路数据选择/分配器对数据I/O线进行组选和字选。

同一组中的寄存器，每次仍只能使用一个。例如要将Z80的寄存器H的内容传送给寄存器D时，就不可能在两个寄存器之间一次直接完成，而必须舍近求远，先选通H，将内容经总线送入暂存器TMP，再选通D，从暂存器取回数据装入。

一行的两个8位数据寄存器构成一个寄存器对，它们常常可以当作一个16位寄存器来使用。但这只不过是指令的简化，事实上执行时仍需由控制器先后选通低8位和高8位寄存器，分次进行操作。

Z80的寄存器组有一种比较独特的“主辅寄存器”结构。一组主寄存器可以和一组辅寄存器一次交换全部内容，而且只需要最短的工作周期。即使它们之间存在着专门的数据通路，要把多达6个字节的数据在不足两个时钟周期之内相互传送到位仍是困难的。其实，简单地利用寻址机构便可达到目的。主、辅寄存器在RAM中各为一组，地址相同，一个T型触发器控制两组的切换。任何时候只有一组可被访问，这时它就是主存贮器，处于后台的是辅助存贮器。执行交换指令时，MPU的内部操作仅是对这个切换触发器触发一次，使其输出翻转，这样就实现了主辅寄存器的地位互换，也就相当于内容互换，实际上无需任何数据移动（图3-14）。在DE和HL寄存器间也应有这样的控制机构，以便它们快速互换内容。

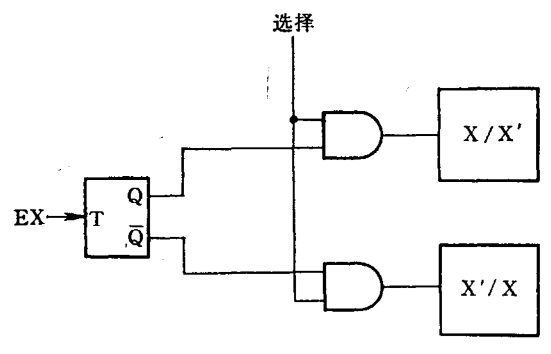


图3-14 主辅寄存器切换控制

3. 地址通道。

寄存器矩阵每一列I/O线除通往数据端口的多路开关，还连接在一个16位的地址锁存器的输入端。一条行线被选中并且地址锁存信

号有效，16 位数据便从地址通道进入锁存器。锁存器的输出经地址总线缓冲器送入地址总线，成为访问内存的地址。

除专用的地址寄存器可向地址通道输出数据，16 位的通用寄存器对也可以向地址通道输出数据——这时它们就成了寻址寄存器。

在寄存器组的地址通道上有一个附属的“增量/减量器”。这是一个 16 位的加 1/减 1 电路，需要递增或递减的地址，由它进行加 1、减 1 操作。16 位寄存器对内容的加 1、减 1 运算，也经由地址通道利用它进行。这种操作若用 8 位 ALU，需要四次传送、两次运算，而采用这种方式可以在基本的取指令周期内完成。这是 Z80 唯一不经过 ALU 的算术运算。

（二）寄存器的分类

下面以 Z80 为例，寄存器组见图 3-15。寄存器的名称，通常以单字符表示 8 位寄存器，双字符表示 16 位寄存器或寄存器对。16 位寄存器的高、低 8 位，以寄存器名加后缀“H”和“L”表示，如 PC_H，PC_L 等。

寄存器控制逻辑	多路开关	
	A	F
	H	L
	D	E
	B	C
	A′	F′
	H′	L′
	D′	E′
	B′	C′
	IX	
	IY	
	SP	
	I	R
	W	Z
	PC	
	增量减量器	
	地址锁存器	

图 3-15　Z80寄存器组

Z80 的寄存器从逻辑结构和用途上可以分为四个部分。

1. 累加器 A 和状态标志寄存器 F。

这两个寄存器上一节已经介绍。它们既是寄存器组的成员，又同运算器有着特殊关系。

它们同其他通用寄存器统一编址使用(寄存器 A 的地址代码是 111,寄存器对 AF 的地址代码是 11)。F 不能作数据寄存器使用,也不提供访问地址代码。A 实际上身兼"累加器 A"和"寄存器A"两个角色,二者在逻辑和控制上都有一定区别。例如,指令 ADD A,A(将A的内容和 A 的内容相加,结果存于 A)。前一个 A 是作为累加器,不必给出寄存器代码,操作码译码后控制器便会打开它同运算器的通道,将数据送入 ACT。而后一个 A 是"寄存器 A",用寄存器代码指定,译码后控制器将它的数据(还是同一内容)经总线送入TMP。

A 和 F 组成 16 位寄存器,只能用于堆栈操作指令 PUSH AF,和 POP AF——这是标志寄存器 F 同总线交换数据的唯一机会。AF 有别于其他寄存器对的另一特点,是它们不能用作寻址寄存器。

A′和F′是辅助寄存器对。用一条指令(EX AF,AF′)便可使它们和 AF 切换,以暂时将 AF 的内容放到"后台"保护起来。

2. 通用寄存器。

指既可作数据寄存器,又可作地址寄存器,其用途可由用户分配的寄存器。Z80 的通用寄存器分为主、辅两组。8 位主寄存器 B,C,D,E,H,L。它们可以单独使用,也可以组成 BC,DE,HL 三个 16 位寄存器对使用。一个寄存器用机器指令中的三位二进制代码指定,即:B-000,C-001,D-010,E-011,H-100,L-101。在某些指令中,通用寄存器对也可用两位二进制代码指定:BC-00,DE-01,HL-10。读者不难从中看出寄存器寻址的某些规律。与寄存器 A(111)和 AF(11)的代码合起来看,唯独没有 110 这个代码。其实指令中是含有代码 110 的,它是数据总线缓冲寄存器的地址码。

通用寄存器中,HL 寄存器在使用上具有一定特殊性。H 表示"高",L 表示"低",是8008 开始有的、连接地址总线高 8 位和低 8 位的寄存器。8080 到 Z80 都把它承袭下来,仍用作主要的寻址寄存器,用它作为扫描内存的指针相当方便,同时 HL 又作"16 位累加器"使用。因此,它是 Z80 中使用频度较高的通用寄存器对。

8 位辅助寄存器组成 BC′,DE′,HL′寄存器对。它们与同名主寄存器对的关系同 AF和 AF′。用一条命令(EXX)便可使三对主、辅寄存器间同时"交换内容",96 位数据的传送只需普通 8 位数据经总线传送的时间。因此,EXX 和 EX AF,AF′合用,可以在响应中断时用来快速"保护现场"——保存当时各寄存器的内容。从中断返回时再交换一次又可复原。因此辅助寄存器又有"内部堆栈"之称。

3. 专用(地址)寄存器。

(1)变址寄存器 IX,IY。它们都可以用来存放一个 16 位的"基地址"。以后只要给出一个 8 位的"偏移量",便能指定基址前后共 256 字节范围内的任一单元,便于数据表的操作。偏移量以 8 位带符号的二进制补码表示,00H～7FH 表示 0～+127,80H～FFH 表示 -128～-1。实际地址需要经过运算器计算得出,所以变址寄存器的内容主要是向数据通道而不是向地址通道输出。只有执行 JP (IX)和 JP (IY)(转入 IX 或 IY 所指入口地址执行)两条指令,才会将它们的内容直接送入地址总线。

(2)堆栈指示器 SP。堆栈是 MPU 在内存存放数据的特殊区域。它像一个开口的容器,口朝着内存低端方向,可由下而上地"堆"放数据,每层一字节,双字节数的高位字节在下。存入或取出数据都只能从最顶端进行,先存入的总在下边,最先取出的总是上面最后存入的,这

叫“后进先出”原则。SP 用来记载堆栈栈顶当前位置(地址),并随着每字节数据的存取而自动增 1 或减 1 。有了这个指针,在堆栈存取数据,只要记住先后顺序,无需过问数据存放的实际地址。

Z80 的堆栈原则上可以设置在寻址范围内的任何 RAM 区。SP 的初值(栈底地址)由指令设定。每次从堆栈取数的操作,是将 SP 内容送到地址总线,然后 SP + 1→SP。每次向堆栈存入数据,要先将 SP − 1 (指向原来栈顶上面一个单元)再送入地址总线,并送回 SP。

(3)中断向量寄存器 I。它用来预先存放中断服务子程序所在的“页数”——地址的高 8 位。在 Z80 的 IM2中断方式下,它的内容送入间接寄存器 W,外部设备发来的低 8 位地址存入间接寄存器 Z,WZ 内容在响应中断后的取指令周期送上地址总线,便可取出中断服务程序的第一条指令。

(4)DRAM 刷新地址寄存器 R。它是唯一的一个 7 位寄存器,D7恒为 0 。其中可以存放一个 7 位的存贮器行地址。在 Z80 取指令周期的后半部分时间,将 R 的内容送入地址总线。DRAM 刷新电路利用它选通一行存贮单元,进行刷新操作。控制器将其增 1 送回,指向下一行。

因为程序中取指令周期的数量是随机的,R 的内容变化也不规则,可以取它的瞬时值作为产生随机数的“种子”。

(5)程序计数器 PC 。它用来存放 MPU 顺序执行指令的存贮器地址指针值。取指令操作时一般由 PC 给出指令操作码的地址。MPU 复位时 PC 内容被置为 0000H ,这也就是 MPU所要执行的第一条指令的地址。PC 的内容每次送入地址锁存器后,都例行增量送回PC,指向程序区的下一个单元。当操作数就跟在操作码之后(如立即寻址等方式)时,读内存地址也由 PC 给出,并同样增量,数据取出后 PC 必然指向下一条指令操作码。转移类指令执行中,可以把新的地址置入 PC ,除此之外别的指令不能改变 PC 的内容。

(6)间接寄存器 WZ 。这对寄存器通常在 Z80 逻辑图和资料上并未给出,因为它们仅供 MPU 内部使用,不能用指令访问。但对于了解 MPU 的工作原理,很有必要认识它们。这里我们是借用 8080 中相同部件的名称。它主要用于地址转换。例如,执行含有直接地址指令 JP NN(转移到地址 NN)的过程中,从存贮器取来目标地址的低 8 位数据后不能立即装入 PC_L 。因为本条指令还需要用原来的 PC 值(已加 1)取出目的地址的高 8 位数据。所以转移目的地址 NN 需要暂存到 WZ 寄存器中。在下一个取指令周期由 WZ 直接将内容送入地址锁存器,作为取指令地址,然后加 1 存入 PC ,以后的指令仍按常规进行。如果没有间接寄存器,执行类似的地址转换指令,中间地址必须占用其他寄存器,改变其内容。这是不允许的。

(三) 寄存器与寻址方式

寻址方式就是提供操作数的方式。寻址方式的多少是 MPU 的重要性能指标。寻址方式愈多,编程就愈加灵活便利。 Z80 有 10 种寻址方式,常使初学者感到眼花缭乱。其实,除“寄存器寻址”、“隐含寻址”是控制器对寄存器的选择,“位寻址”是控制器对运算器的操作选择外,其余的七种方式就是把不同的寄存器内容直接或间接地经地址锁存器 ADR 送上地址总线。现在我们从寄存器组的角度来分析各种寻址方式的机制。

1. 立即寻址。 PC→ADR。

例如指令LD　C，FFH(将数FFH送入寄存器C)。执行时将程序计数器PC当前内容(取出指令操作码后PC已经加1，正好是存放FFH的单元地址)送上地址总线，即取得操作数。

2．扩展立即寻址。 PC→ADR,PC+1→ADR。

例如指令LD　BC,F000H(将16位数F000H送入BC寄存器对)。执行时分两步将PC内容(每次并加1)送上地址总线，将数据00H取入C，F0H取入B。

3．寄存器对间接寻址。 寄存器对(HL、BC、DE、SP、IX、IY)→ADR。

例如指令LD　(DE),A(A中数据存入以DE内容为地址的内存单元)。执行时将寄存器对DE内容送地址总线，然后送出A中数据，存入选中的内存单元。

堆栈寻址的操作是：SP(或SP－1)→ADR,SP＋1(或SP－1)→ADR。

如指令PUSH　AF(将AF的内容存入堆栈)。执行时先将SP内容减1，送上地址总线，存A；再将SP减1送上地址总线，存F。

4．直接寻址。 操作数地址→WZ→ADR。

例如指令LD　A,(8000H)(将8000H单元所存数据送入寄存器A)。执行时先将操作数地址8000H分两次取入间接寄存器Z和W，再将WZ内容送上地址总线，取得目标数据。

5．零页寻址。 地址码转换→WZ→ADR。

例如RST　10H(11010111——调用以地址00010000为入口的子程序)。地址码是三位二进制数*nnn*，表示内存零页(高8位00H)中低8位为00*nnn*000的地址。执行时先将W清0，操作码的其他各位置0后存入Z，然后将WZ送上地址总线，作为取指令地址。

6．相对寻址。 PC＋偏移量→WZ→ADR。

例如指令JR　C0H(跳转到距当前执行地址C0H——即－64个单元的指令)。执行时先取入偏移量*e*，$PC_L+e\rightarrow Z\cdot CY$，$PC_H+CY\rightarrow W$，WZ送上地址总线(新的指令地址)。

7．变址寻址。 IX(IY)＋偏移量→WZ→ADR。

例如指令ADD　A(IX＋d)　(将IX＋d单元的数据与A相加并存入A)。操作数d也是一个偏移量，取入后用同样方法和变址寄存器IX所存的基地址相加并存于WZ，再由WZ送上地址总线。

四、控　制　器

(一)　周期和时序

为了分析MPU内部的控制逻辑，先观察一下它的工作方式。MPU的工作具有周期性、顺序性和节拍性的特点。

1．周期。

MPU的全部工作，就是执行预先存放在存贮器里的机器指令序列——程序。对于每一条指令，都要经历“取指令”和“执行指令”两个步骤。执行程序的全过程就是这两个步骤的周期性循环。每一次循环称为一个“指令周期”。

早期电子计算机由于使用的磁芯存贮器工作速度较低，而且信息的读出是破坏性的，每次读数据后还必须重新写入。从时间开销上看，访问存贮器成了CPU的主要工作。因而把访

问存贮器一次的过程定义为一个“机器周期”（M 周期）。取指令必须访问内存，是一个机器周期。执行指令可分为两部分。CPU 对指令译码以及执行规定的运算需时很少，在取指令周期后部对存贮器重写时便可同时完成。执行指令一般都需要访问一次存贮器，以取得操作数或存放运算结果，所以也要用一个机器周期。每个机器周期的工作过程，是内部逻辑部件按一定顺序进行的若干“微操作”的组合。各种机器周期的微操作种类和步数不一，但对于时钟同步式计算机，每步微操作经历的时间却是相同的——都相当于 CPU 所用时钟脉冲的一个周期，称为“时钟周期”（T 周期）。

早期计算机指令格式比较一致，每个指令周期固定地包含“取指令”和“执行指令”两个机器周期。每个机器周期又包含着固定个数的时钟周期。实现这种固定周期方式的控制逻辑比较简易，一个时标系统周而复始地顺序发出同样一组节拍信号——这很像做体操时的“一、二、三、四，二、二、三、四……”口令，便能循环控制每条指令的取入和执行。固定长度的周期必需满足微操作步数最多的指令对时间的要求，在执行需时较短的指令时便会出现无所事事的“空操作”节拍，造成机器时间的浪费。

微处理器的出现和半导体存贮器的应用，改变了上述的 CPU 工作条件。指令格式的多样化，存贮器的高速度，带来了可变周期方式。例如 Z80，指令长度可为 1～4 字节，指令的微操作数量相差也很大。虽然指令周期、机器周期的定义不变，周期性依然存在，但一个指令周期含有的机器周期数和一个机器周期含有的时钟周期数却是很不规则的。可以说，MPU 有多少种指令，指令周期的组成就有多少种变化。

Z80 的一个指令周期，根据指令的内容，可由 1～6 个机器周期组成。第一个机器周期必然是从内存取出指令的操作码，称为“取指周期”（M_1）。指令的操作码为一个以上时，对于每个字节都需用一个 M_1 周期依次取出。然后根据指令的需要，一般可有一至二个内存或 I/O 口读写周期。如果指令只涉及 MPU 内部的简单操作，例如 LD B，C（将寄存器 C 的内容送入 B），在取指周期后半部对 DRAM 进行刷新时就能完成，这个指令周期就只有一个 M_1 周期而没有内存读写周期。

Z80 的机器周期中，取指周期通常由四个 T 周期，内存读写周期通常由三个 T 周期组成。当取指令和读数据后需进行比较复杂的运算时，则在机器周期中增加一个或几个时钟周期来完成操作。所以机器周期的长度可为 3～6 个 T 周期。有一些指令的内部操作比较复杂，不能在访存周期中附带完成，指令周期中就还要增加专门的（不访问存贮器的）内部操作机器周期。

以上都是常规的机器周期。当低速外部器件向 MPU 的 $\overline{\text{WAIT}}$ 控制线发来低电平（“请等待”）信号，MPU 检测发现后将自动在机器周期中插入等待周期 T_W。此外，当 MPU 响应外部器件的总线请求、中断请求和执行暂停指令后，也将经历特殊的机器周期。

2. 时序。

MPU 的各种机器周期中，各步微操作是按一定的时间顺序进行的。这称为“操作时序”。从外部看，各引脚的输出信号随内部操作时序而呈有规律的变化，按一定的时间和顺序向外部器件发出各种信号，同时也按一定的时序接收外部器件的信号。

下面以一个指令周期的实例来认识 MPU 的时序。Z80 指令 INC（HL）（将 HL 寄存器对指定的内存单元内容加 1）是一个很好的例子。因为 MPU 并不能直接对内存单元实施加

1操作，必须先将目标单元内容取入，加1后再行送还。所以这条貌似简单的指令竟需要三个机器周期、共11个T周期方能完成。

假若我们在MPU的各引脚接上具有信息存贮和低速重现功能的示波器，便能观察到指令执行全过程中输出信号的时序变化(图3-16)。顶上的一列是时钟信号ϕ的波形，可以作为一个参照的时刻表。用它作对照，我们将会发现每个时钟周期的微操作并不都发生于同一时刻。有的发生于时钟脉冲的上升沿后，有的发生在下降沿后。也就是说，这个“时钟”在上升沿和下降沿各“敲响”一次。因此按“钟声”进行的操作之间可有半个周期的时间差。下面我们用“↑”表示时钟脉冲上升沿，“↓”表示下降沿。

M1　(取指周期)　这个周期是在上一条指令执行结束后自动进入的。

T1↑　$\overline{M1}$信号首先有效，表示进入取指周期。随即，程序计数器PC当前的内容(正是本条指令存放的地址)送上地址总线。

↓　$\overline{MREQ}$、$\overline{RD}$信号变得有效，作为目标芯片选通和读出数据的控制信号。

T2↑　PC+1→PC，指向下一单元地址。

↓　检测$\overline{WAIT}$信号是否有效，有效时将插入T_w周期。

T3↑　将此刻数据总线上的数据(应为程序存贮器送出的指令操作码)取入指令寄存器(接着译码)，并使$\overline{M1}$、$\overline{MREQ}$、$\overline{RD}$无效。$\overline{RFSH}$有效，DRAM刷新寄存器R的内容(待刷新单元行地址)送入地址总线。

↓　$\overline{MREQ}$再次变低，配合$\overline{RFSH}$选通所有待刷新单元，实现刷新操作，此时$\overline{RD}$无效，存贮器数据不会进入数据总线。

T4↑　R+1→R指向待刷新的下一行地址。

↓　$\overline{MREQ}$无效，刷新阶段结束。$\overline{RFSH}$将在下个T周期开始时无效。

M2　(读内存周期)　这是对INC (HL)指令译码后产生的一组执行操作。

T1↑　HL寄存器对内容(操作数地址)送入地址总线。

↓　$\overline{MREQ}$、$\overline{RD}$变低以启动内存。

T2↑　等待内存送出数据。

↓　检测$\overline{WAIT}$信号。

T3↑　等待内存送出数据。

↓　将数据总线上的操作数(应已在总线上)取入暂存器TMP，$\overline{MREQ}$、$\overline{RD}$无效。

T4　基本的读内存周期到T3结束，但本指令还需进行加1运算，T3剩下的时间不够，因而增加了T4周期，用于对TMP的内容进行加1运算。

M3　(写内存周期)　实际是M2操作的后续部分，也是指令译码所产生。

T1↑　地址总线上再次送出HL中的地址。

↓　$\overline{MREQ}$有效，ALU的运算结果送入数据总线缓冲锁存器。

T2↑　等待写入内存。

↓　检测$\overline{WAIT}$信号，$\overline{WR}$信号有效，作为存贮器的写入脉冲。

T3↑　等待写入内存。

↓　(数据应已写入)$\overline{MREQ}$、$\overline{WR}$信号无效，写操作结束，启动下一个取指周期。

以上三种机器周期，是MPU最基本的时序。Z80除此之外还有I/O口读写、总线请求响

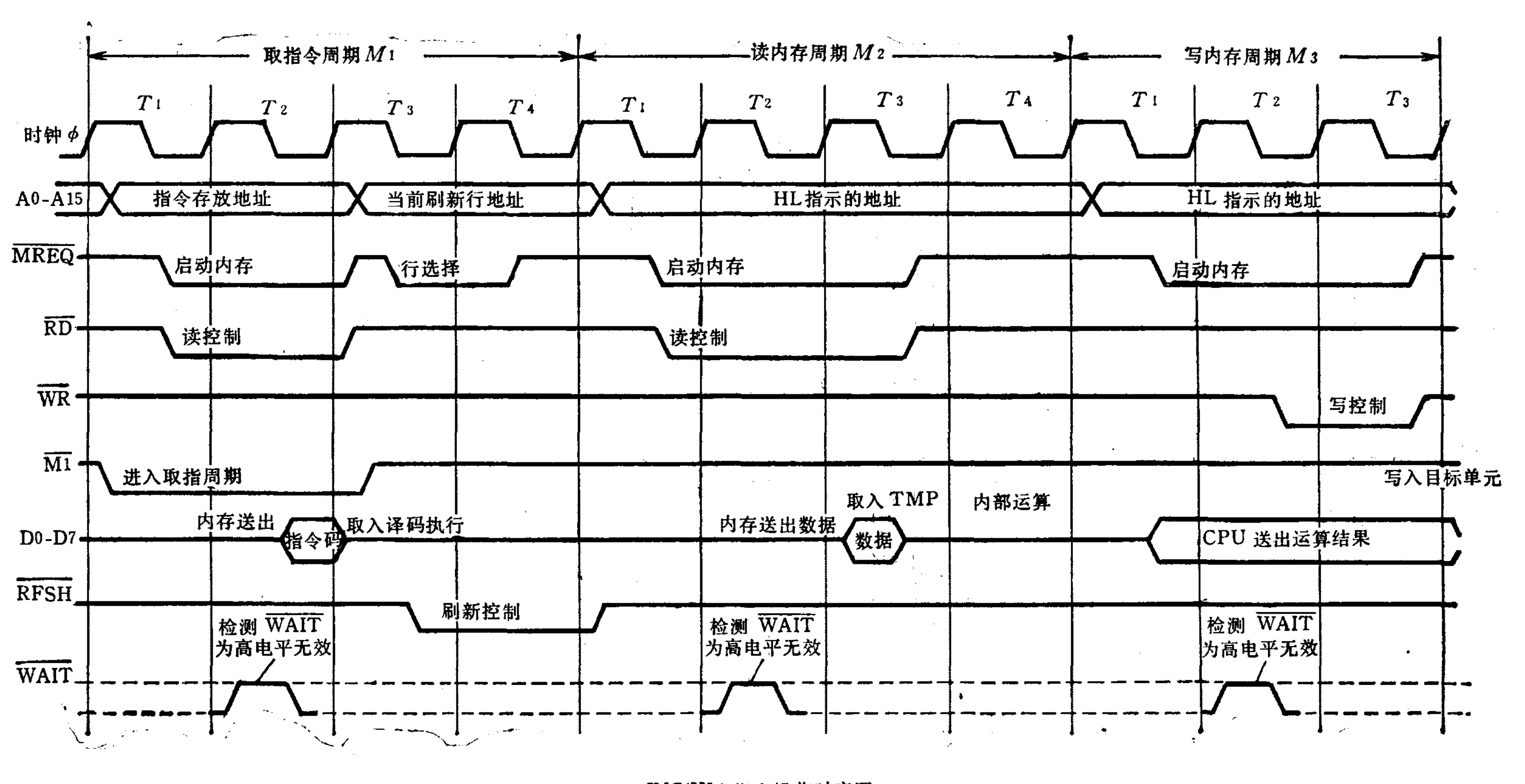


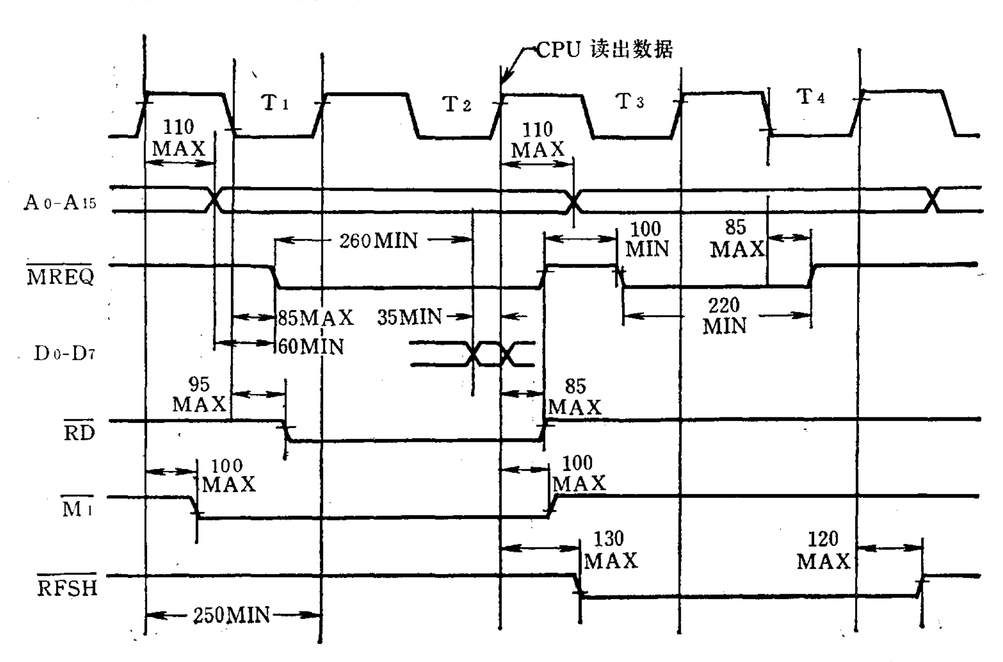
图 3-16 INC(HL) 指令操作时序图

应、不可屏蔽中断请求响应、可屏蔽中断请求响应和从暂停状态退出等独特的机器周期时序。

上面仅是对 MPU 时序的粗略描述。实际上每一步都有相当精确的定时。生产厂商公布的技术资料中列有交流特性参数，包括每种信号变化的时间尺度，供用户建立计算机系统时参考。

确切地掌握 MPU 的时序，对于系统的配置是非常必要的。因为每种逻辑器件由于材料、结构和工艺的关系，对数字信号的接收、转化、传输都有一定的时序。将它们联系起来组成系统，相互的时序必须适配。例如 MPU 向存贮器取数，首先由 MPU 向对方发去地址和有关控制信号。存贮器接收到这些信号后作出响应，将指定单元的内容送上数据总线。它响应的速度是由存贮器件的性能、而不是 MPU 的性能决定的，有快有慢。然而一般情况下 MPU 并不"了解"对方数据是否送出，总是按上述的固定时刻表"盲目"从数据总线上取入数据。如果这时存贮器还未来得及将数据送出，MPU 取得的就是错误的数据了。

这里以 Z80A 的取指周期为例说明时序配合问题。图 3-17 是 Z80A 取指周期定时图。时间以毫微秒(nm)为单位，"MAX"表示"最大"，"MIN"表示"最小"。需要说明的是，MPU 的交流参数同负载轻重也有关系。图上所标的参数是在 50pF 负载条件下测得。当负载增多，电容增大时，由于充放电时间延长，电平转变的延迟也会增加。



(时间单位：毫微秒(nm))

图 3-17　Z80取指周期定时图

Z80A 时钟频率 4 MHz 时，每个时钟周期 250nm 。在取指周期，T1 上升沿到地址在总线上出现的延迟时间为 110nm，数据总线上的数据建立时间应比 T3 上升沿提前 35nm 。从地址有效到数据建立的时间 $Trd = T1 - 110 + T2 - 35 = 355nm$。由于 MPU 的地址信号还要经过译码器、多路转换器等逻辑器件的延时才能到达存贮器件，所以要求程序存贮器在收到地

址信号后必须在远小于 355nm 的时间内将数据送出，否则 MPU 届时就取不到指令代码。

存贮器的性能参数中，从地址有效到数据送出的时间称为"取数时间"，是器件的重要性能指标。同 Z80A 配接的程序存贮器取数时间必需≪355nm。

随着 IC 生产技术的迅速进步，现在市场上的存贮器件工作速度一般都能满足上述要求。生产厂商常将存取时间作为存贮器件成品分档指标，并在型号后标明档次。例如，6264-15 ($T_{AAC}\leqslant$150nm)，4116-7 ($T_{RAC}\leqslant$70nm)，等等。

计算机的逻辑部件必须同 CPU 的各个机器周期时序都能适配。必须使用低速器件时，解决的办法是增设$\overline{WAIT}$信号发生器，使CPU插入等待周期(T_W)。

（二） 控制逻辑

1. 控制信号。

迄今我们已经认识了 MPU 的主要执行部件——运算器、寄存器组、总线及其接口电路。把它们有机地组合起来，就能完成 MPU 的基本操作——数据的取入、运算、暂存、内部传送和输出。

计算机可以比喻为一座自动化工厂。它的生产原料——数据是存放在自动化仓库——存贮器中的，用地址信号就能经传送带调出使用。MPU 的数据通道好似一条生产流水线。运算器是多通道、多功能的加工机，原料从不同的通道经过时便受到不同的加工处理。寄存器是备料间和半成品暂存处；内部数据总线是传送带，输入/输出接口是生产线的进料口和成品输出口。这些逻辑部件通电后就进入了工作状态，并不需要对它们的启动或停止施加控制。需要控制的只是原料从哪里取出，经由什么路径或加工流程，最后又送入何处。也就是说，MPU 的各种操作，就是通过对数据通道内信息流动的控制来实现的。

数据通道各个部分的出入口和通道的分支点，设有相当于"闸门"、"道岔"的门电路。每个门电路的控制线都通往相当于工厂中央控制台的 MPU 控制器。控制器按照程序指令的要求，发出一系列控制信号到各个控制点，控制这些门电路的开启、关闭或切换，使数据按需要在各寄存器、运算器和内外总线间，以及在运算器内的流动，从而实现指令规定的操作。

由前面的讨论已经看到，为使各执行部件按要求工作，需要很多控制信号。例如：寄存器(及输入/输出接口)的选择信号和读/写信号；各部件驱动总线的允许信号；运算器的运算选择和各种控制信号；控制总线触发器置位/复位信号……

一个功能比较完善的 MPU，分布在各控制点上的控制信号可达百个以上。它们都源出于控制器。

2. 指令控制逻辑。

控制器是一个由多位输入产生相应的多位输出的逻辑部件，因而从总体上看是一个译码-编码器结构。控制器的主要输入信息是由存贮器取来的指令。与阵列的指令译码器对指令译码，再驱动或阵列的编码器输出执行该指令的全部控制信号。

除指令外，外电路送入的 CPU 控制信号和 $\overline{WAIT}$ 信号，检测获得的运算器状态信息，也是控制器的输入，能够在一定程度上改变控制器的工作和输出；此外，还需要时序控制逻辑。为了便于分析，这些均先不考虑。

控制器基本结构原理见图 3-18。假设执行指令码 01H 需向控制点 A、C、P、R ……发出有效信号，执行 85H 需向 A、D、P……，执行 FEH 需向 B、D、P、R……各点发出有效信号，则

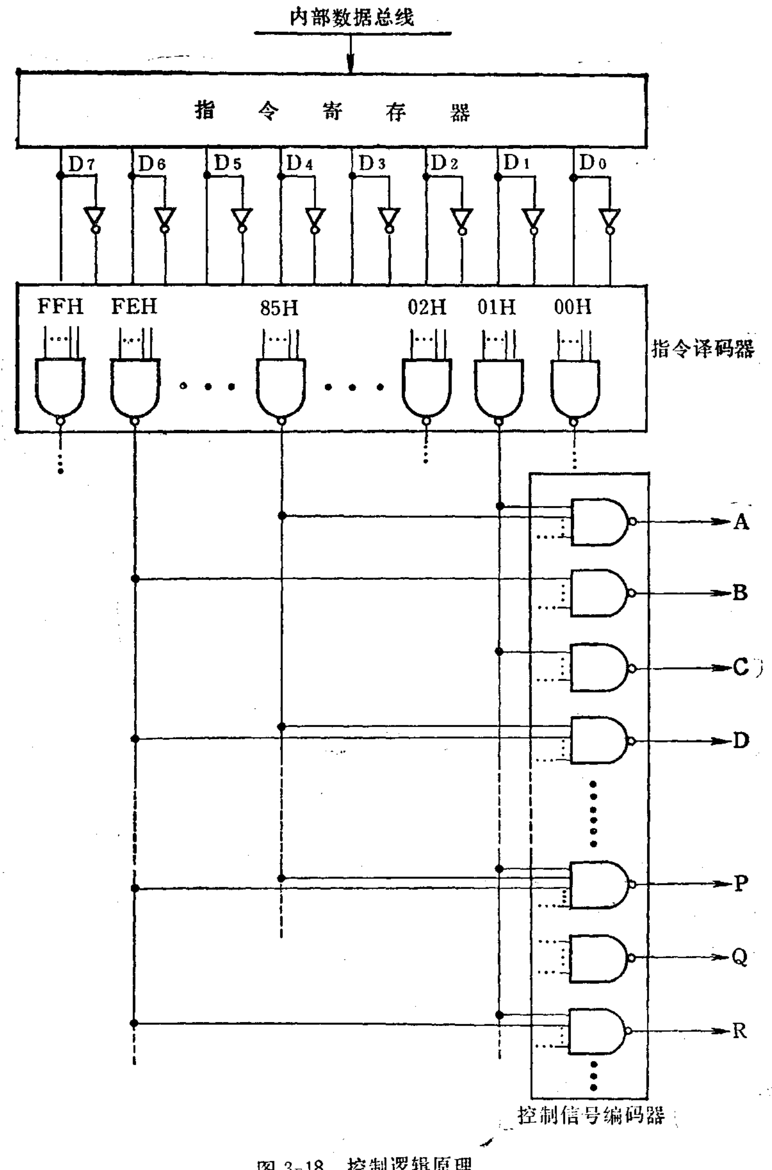


图 3-18 控制逻辑原理

控制逻辑如图所示。图中将电路简化为一级与逻辑和一级或逻辑。指令在取指周期由数据总线取入指令寄存器。指令寄存器的每位输出分为两路，其中一路经反相，变成互反的两个输入信号分配到译码器的各个与非门。当 MPU 指令字长 8 位时，可有 256 种不同的指令字，译码器也就需用 256 个与非门。每当指令寄存器装入一个指令字后，必定有一个与非门输入变为全 1，因而输出为 0（负逻辑有效信号），其余与非门输出均为无效。

MPU 内部加在每个控制点的控制信号，都由编码器的一位——一个负逻辑或非门（即正逻辑与非门）给出。这些与非门有多个输入端，分别连接着与本控制点有关指令的译码输出线。一条指令的执行需要哪些控制信号有效，它的译码输出就通向控制它们的与非门。指令

译码输出的负逻辑有效信号使连接的各个与非门输出负都为1——即正逻辑有效信号，以实现所需的控制。

3．时序逻辑。

我们已经知道，MPU的工作方式具有时序性。执行一条指令时，各逻辑部件的工作有的是同时进行的，有的却必须依照严格的先后顺序，分步进行。上述单纯体现指令控制逻辑的电路，指令字一次译码产生出所需的全部控制信号，并不能使指令得以正确执行。怎样使这些控制信号不致同时加到控制点，而是按周期时序分批发生作用呢？这就需要加入时序控制逻辑。

时序控制的方法随总的控制逻辑类型而异。这里先介绍"时标"方式。

MPU的操作(即控制信号的分批发出)要按一定的"时间表"进行，必须得有一个能准确指示时刻的"钟"。MPU输入的"时钟"脉冲信号，只相当于钟表摆轮的摆动，虽有精确的时间间隔，但并不能表示确切的时刻。钟表作为计时器，是用机械方式对摆轮的摆动次数计数，计数值通过指针和刻度标识出时、分、秒来。与此原理相似，MPU内部也可以用一个对基准时钟脉冲计数的计数器，来作为时序控制的"钟"。

假设MPU每个指令周期分为两个机器周期，每个机器周期分为4个时钟周期，就可用一个3位二进制计数器作为时钟，用D1D0的计数值表示"第几个时钟周期"，用D2表示"第几个机器周期"。这称为"时标"，类似于钟表的"秒"数和"分"数。随时钟脉冲计数增值，计数器达到最大值111后再来一个脉冲又恢复为000。实际使用的时标系统不一定这样规则，因而更复杂一些，需在脉冲计数基础上再通过译码产生。

因为取入每条指令的取指周期操作都是相同的，而且不需要操作码参与，可直接用时标信号控制操作。将D2输出反相接入四个与非门，当D2＝0时，由D1D0的计数值00～11顺序发出取指周期四个时钟周期的控制信号，将PC指示的当前指令取入指令寄存器IR。

指令寄存器的输出与时标信号同时作为指令译码器的输入(图3-19)。在执行部分，因D2

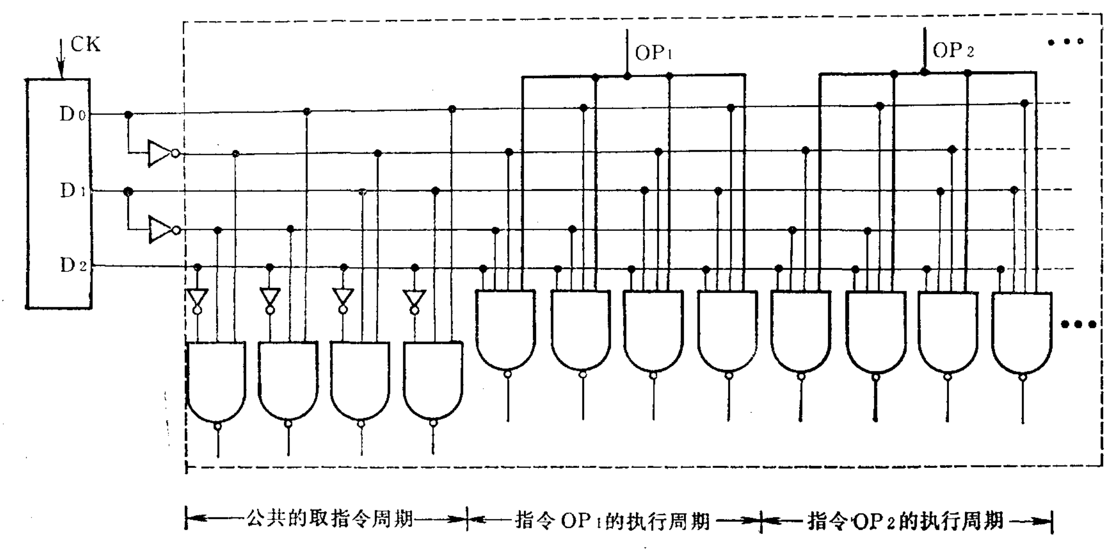


图3-19　加上时标后的译码器

的输出不取反值，D2＝0时译码器不工作，由上述取指逻辑控制。增加时标后译码器的与非门数需增加4倍。这样，每一指令执行周期的操作便可分为四步。指令代码为OP时，用OP·100控制M2T1的微操作，OP·101控制M2T2的微操作……。在指令寄存器中代码不变的情况下，随着时钟的“走动”，便可以顺序发出4批控制信号，完成执行周期。下一个时钟脉冲到来，时钟计数器输出000，便进入下一个取指周期。

这种单级控制方式的主要问题，是逻辑门的扇入扇出系数过大，指令寄存器和译码器每个输出端都要驱动大量负载，编码器的某些与非门也有很多路输入。增加控制器的级数，可以带来结构的简化。将指令码分段或控制信号分组，都可以以增加级数来减少扇入扇出系数。在指令译码后再加时标，也是一种常用的方式。

以上的控制方式比较适用于固定周期的MPU。若指令周期的长度不等，时标系统必须按最长的周期来设置。这样在执行短周期指令时会产生一些空操作的时钟周期。对此，可以在执行周期的最后产生一个“结束”信号反馈给控制器或时标发生器，强制回到取指周期。

4. 同步控制和异步控制。

以上控制方式称为“同步控制”，微操作按时钟信号的节奏分步进行。采用时钟同步方式有利于简化控制逻辑。生产流水线设计时便使每个工位能在同一时间单位内完成自己的操作，时间一到就可将工件移送下一工序。那么，管理者就不必再去过问每一工序的工作情况，处理工序衔接问题。只需按时刻表一次次发出信号，驱动传送带运转，成品自会源源流出。

时钟同步的MPU，允许使用的最高时钟频率受MPU工艺和结构的限制。因为一次微操作，从控制信号的产生到操作完成需经过很多逻辑门的延迟，需要一定的时间才能完成。MPU通常按要求最长时钟周期的微操作来确定频率上限，并以此作为产品分档指标。如Z80为2.5 MHz，Z80A为4MHz，Z80B为6MHz。一般可适当降频使用。这样，MPU实际的工作速度就由时钟频率决定，在额定上限以下，时钟频率用得越高，工作速度就越快。

由于每道工序所需的时间不可能完全相等，按照最费时工序的需要而确定的时间单位，对其他工序必然造成时间的浪费。若将时钟驱动改为“事件驱动”，就成为“异步控制”方式。异步控制不用时钟来统一步调，每一工序操作完成时就发出一个控制信号，立即启动下一工序操作。这样可以保证每一工序在时间上都达到满负荷，自然可以提高工作效率，但控制逻辑复杂得多。所以多年以来，提高MPU速度的努力主要仍放在改进IC工艺和材料方面，现代最高速的MPU，依然沿用会“浪费”时间的时钟同步方式。

（三） 微程序控制器

用上述组合逻辑电路构成的控制器，主要通过逻辑门间的硬连线建立。因为输入和输出的逻辑关系是随机的，每一条连线都需要单独设计。当指令种类较多时，控制器电路非常复杂，不仅硬件设计工作量大，而且一旦定型后再要局部改动都必须重新布线。对于MPU的生产工艺来说，这种不规则的电路结构导致了杂乱的掩模图形，也增加了图版设计的难度。

因此，还在计算机发展早期的50年代初，英国剑桥大学的威尔克斯就提出了一种“微程序控制”的思想。因为控制器产生的一组控制信号可以看作一个多位的数据（控制字），那么先把这些数据有序地存放在存贮器中，需要取出送到各控制点，作用是相同的。

1. 微程序控制方式。

威尔克斯的设想是，将控制系统作为一个存贮矩阵。矩阵的一行对应于一个时钟周期，每一列对应于一条控制线。一条指令各个时钟周期的控制数据按行地址顺序存放。在一行的各个存贮单元中，本时钟周期内有效的控制位存 1，其余位都存 0，组成一个控制字。

这样的存贮器称为“控制存贮器”。MPU的指令代码，就作为控制存贮器的地址码。经过译码器译码后，可产生控制存贮器的行选信号，将存贮的控制字变成控制信号输出。

威尔克斯的方法和通常的计算机程序控制非常相似，很多地方需要相同的逻辑结构和技巧。因此，他借用了一些程序设计的术语来描述他的系统，不过给它们加上前缀“micro”（“微”）；而常规术语则常加上前缀“macro”（“宏”），以便区别。例如前面已讲到的“微操作”，表示微操作的控制码称为“微命令”，控制矩阵每一行的微代码组合称为一条“微指令”，具有一定功能的连续微指令序列称为一个“微程序”，等等。

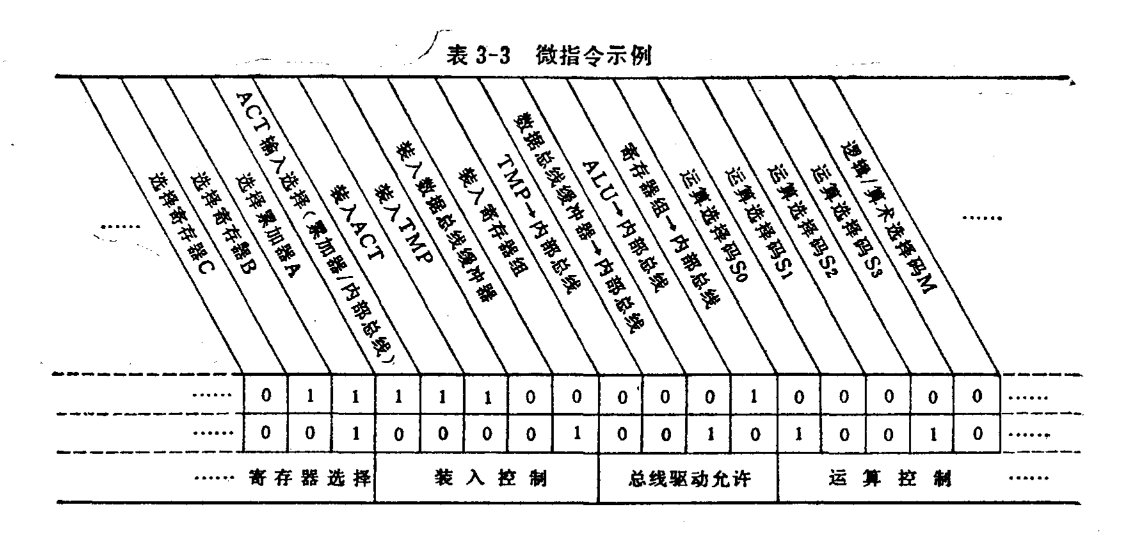
一条微指令（即控制存贮器的一行微码）原则上应含有涉及 MPU 所有控制点的微命令，因而字长是相当可观的。以执行宏指令 LD A，B 的微程序中运算操作的两条微指令为例，可用助记符表达为：

$A \longrightarrow ACT, B \longrightarrow TMP$

$(ACT) + (TMP) \longrightarrow A$

应用前面的分析，可以给出这两条微指令的主要部分如表 3-3 所示，表中省略了其余的微代码。

表 3-3 微指令示例



……	选择寄存器C	选择寄存器B	选择累加器A	ACT输入选择（累加器/内部总线）	装入ACT	装入TMP	装入数据总线缓冲器	装入寄存器组	TMP→内部总线	数据总线缓冲器→内部总线	ALU→内部总线	寄存器组→内部总线	运算选择码S0	运算选择码S1	运算选择码S2	运算选择码S3	逻辑/算术选择码M	……
……	0	1	1	1	1	1	0	0	0	0	0	1	0	0	0	0	0	……
……	0	0	1	0	0	0	0	1	0	0	1	0	1	0	0	1	0	……
……	寄存器选择			装入控制					总线驱动允许				运算控制					……

从微指令的内容可以看出，将前一条微指令的各位分别送上各条控制线，就能将 A 的内容送入 ACT，B 的内容经内部总线送入 TMP。发出后一条微指令，则将 ACT 和 TMP 的内容同时送入 ALU 进行加运算，并经内部总线送入 A，实现了所需的控制逻辑功能。

采取微程序方式，把控制器的逻辑设计由硬件工作变成了软件工作。复杂的随机布线，变成了在固定的阵列中填入“1”或“0”。在微程序设计完成以后，制作 IC 的掩模设计与普通 ROM 无异。修改控制存贮器的内容，只是把某些位的数据在 0 和 1 间改变一下，而无需对整个电路“推倒重来”。如果控制存贮器采用可重写的 PROM，还可提供用户自行设计

MPU 指令系统的条件。

但是，这些明显的好处并未使威尔克斯模式很快得到实现。因为他的思想超越了当时的物质基础——那时还没有适于作控制存贮器使用的高速廉价的存贮器件。若用低速存贮器件充当，CPU的工作速度将远比组合逻辑控制方式为低。

直到十多年后，IBM 公司为了解决它的系列计算机程序兼容性问题，花了很大的投资将微程序设计付诸实施，1964 年研制出采用微程序控制的 7 种具有软件兼容性的计算机。到了 70 年代，由于存贮器技术的进步，微程序控制已成为计算机控制器的主要模式。只有那些对速度要求特别高的超级计算机，才仍旧采用组合逻辑方式。

2. MPU 里的计算机。

将指令代码作为一个地址(或间接地址)，所对应的就不是一条微指令，而是一段微程序。为了执行这段微程序，可设置一个类似于程序计数器 PC 的“微程序计数器 (μPC)”。以它的并行输出作为地址译码器的地址输入。指令代码变成一个微程序的首地址，执行第一个时钟周期的微指令后，微程序计数器内容增 1，就指向下一条微指令的地址。取宏指令的微程序是相对独立的，一个执行宏指令的微程序的最后一条微指令，应含有回到取指微程序的控制信号。

至此，读者可能已经发现，这样的微程序控制器很像一个“微微计算机”。微程序集合就像计算机 ROM 里的“解释程序”，用来“解释执行” 机器指令。微处理器本是为控制计算机而研制的，但归根结底却要由一个“计算机”来控制。控制器本来是为执行软件而设置的硬件，现在却基本上成为一个软件系统。这不能不说是一个发人深思的计算机哲学问题。从硬件的角度来看，微型计算机—微处理器—微程序控制器是三个层次的相似结构。从软件的角度看，高级或汇编语言—机器语言—微代码语言又是三个相似的层次。它们相互渗透，互为表里，表现了事物对立统一，量变质变和螺旋发展的辩证规律。

这种辩证关系推动着微程序技术的发展。MPU 的设计中，微程序的编制仍然是一件耗时费力的工作。因为微指令的字长(即控制信号的数目)远比宏指令大。一个 8 位的宏指令系统可有 $2^8=256$ 种指令，而一个 50 位的微指令系统则可能有 2^{50} 种指令。这可是一个天文数字！因此，科技人员已仿效宏编译程序研制出高级微程序设计语言，提高了编程效率。另外，正像微型计算机允许用户使用自己研制的系统程序一样，也产生了可由用户设计微程序控制的微处理器。当控制存贮器采用 PROM 结构并提供写入手段时，你就可以建立自己的 MPU 指令系统了。

现在我们已经明白，计算机执行高级语言程序，先要将它转换为机器语言指令送入MPU，而 MPU 又要将它转换为微程序代码，才开始真正的执行。而每次转换中都必须经过很多并非每次都必须的公共过程，造成时间的浪费。如果能够简化层次，显然可以提高工作速度。例如用 ALU 的加法器作乘除运算，通常是用一段含有很多条加减、移位和计数循环指令的程序来完成。如果将执行这些宏指令的微操作组合成一个微程序，用一条乘或除的宏指令执行这个微程序，便可以省掉多次取指令和指令译码的时间，显著提高执行速度。按照这种思想，研究人员开发了直接将高级语言翻译成微代码的系统软件，在同等的硬件环境下，执行同样任务的速度可提高 2～4 倍。

(四) 控制存贮器

1. 卧式结构和立式结构。

如果 MPU 内部的所有控制点都由控制器直接控制，每个点都连接着控制存贮器的一条列输出线，执行每条微指令读出的信息直接加到控制点上，即实现微操作。这种方式使用的电路简单，速度也快。但是 MPU 的控制点很多，如果有 100 个控制点，控制存贮器的字长就必须有 100 位，形成一个行幅度很宽的卧式阵列。

实际上，每一次微操作涉及的控制点都只占 MPU 全部控制点中的少数。所以卧式结构的存贮矩阵的每一行(一条微指令)，只有少数位是"1"，多数位都是"0"，"有效"信号被"无效"信号淹没了，对可贵的存贮器面积的利用率很低。

逻辑设计时，增加逻辑电路的级数往往可以减少其并行的路数。如果将控制代码进行编码存贮，然后再译码还原，就能使微指令得到"浓缩"。有很多控制信号是"势不两立"的，不可能同时有效。假如内部总线上挂有 8 个总线驱动器，8 个三态输出允许信号中任何时候都只有一个有效，所以在控制矩阵中就可以只用一组 3 位微代码来表示这 8 个信号。在三条列线的输出端，用一个 3-8 译码电路便能将每个总线驱动器的控制信号分离出来。运算器的控制信号较多，将每种运算操作用一个运算选择代码定义，再经 ALU 控制逻辑电路译码并编码产生出来，一个三位的微代码便能对主要的算术运算和逻辑运算进行选择控制了。这样的微代码组合称为"字段"。

用最大编码的方法可以大大缩短微指令的字长，使控制矩阵变成高而窄的形状，所以称为"立式"结构。立式结构的控制器需要存贮空间比卧式结构少得多，但每个字段的编码输出都需再经译码器译码，因而增加了时间的延迟，这是它的缺点。实际的控制存贮器多为介于立式与卧式之间的折衷类型。

2. ROM 和 PLA。

用 ROM 作控制存贮器，存贮矩阵是可编程的，通过掩模能够任意对每一位写 1 或 0 。而它的地址译码器是不可编程的。它按输入的地址信号产生它们的全部最小项函数，即对于 n 位输入产生 2^n 个输出。例如一个八位的地址译码器，有 256 组输入数据和 256 个输出端。每个输入数据选通存贮矩阵中唯一的一行字线。这在应用时可能出现两个问题：第一，译码地址并不一定是连续的和穷尽的，因而译码器利用率不高。第二，由于地址的专一性，各个微程序中含有的完全相同的微指令也必须重复存贮，分别读取。

从这个角度看来，可编程逻辑阵列——PLA 对ROM显出一定的优越性。PLA 是在ROM技术的基础上发展起来的。它由一个与阵列 ROM 和一个或阵列 ROM 联结构成。与阵列和或阵列都是可以编程的。

PLA 的结构原理见图 3-20。图中每条行、列线的交叉点上都做有二极管，是否连线则由编程决定。为了清晰易看，图中只画出联接的管子。对于每条列线来说，与阵列的二极管构成与电路，必须有关的行线都为高电平这条列线才能保持高电平，任一行线电平为低就能通过二极管把该列电平拉低。而对于或阵列的各行，二极管则构成或电路，只要任一列为高电平，就能通过二极管使该行线电平升高。

实用的 PLA 多由 NMOS 管组成，同上述结构仅在电气性能和极性上有差别，逻辑原理相同。

PLA 的与阵列可用作相当于译码器的"地址阵列"，或阵列则用作相当于存贮器的"数据

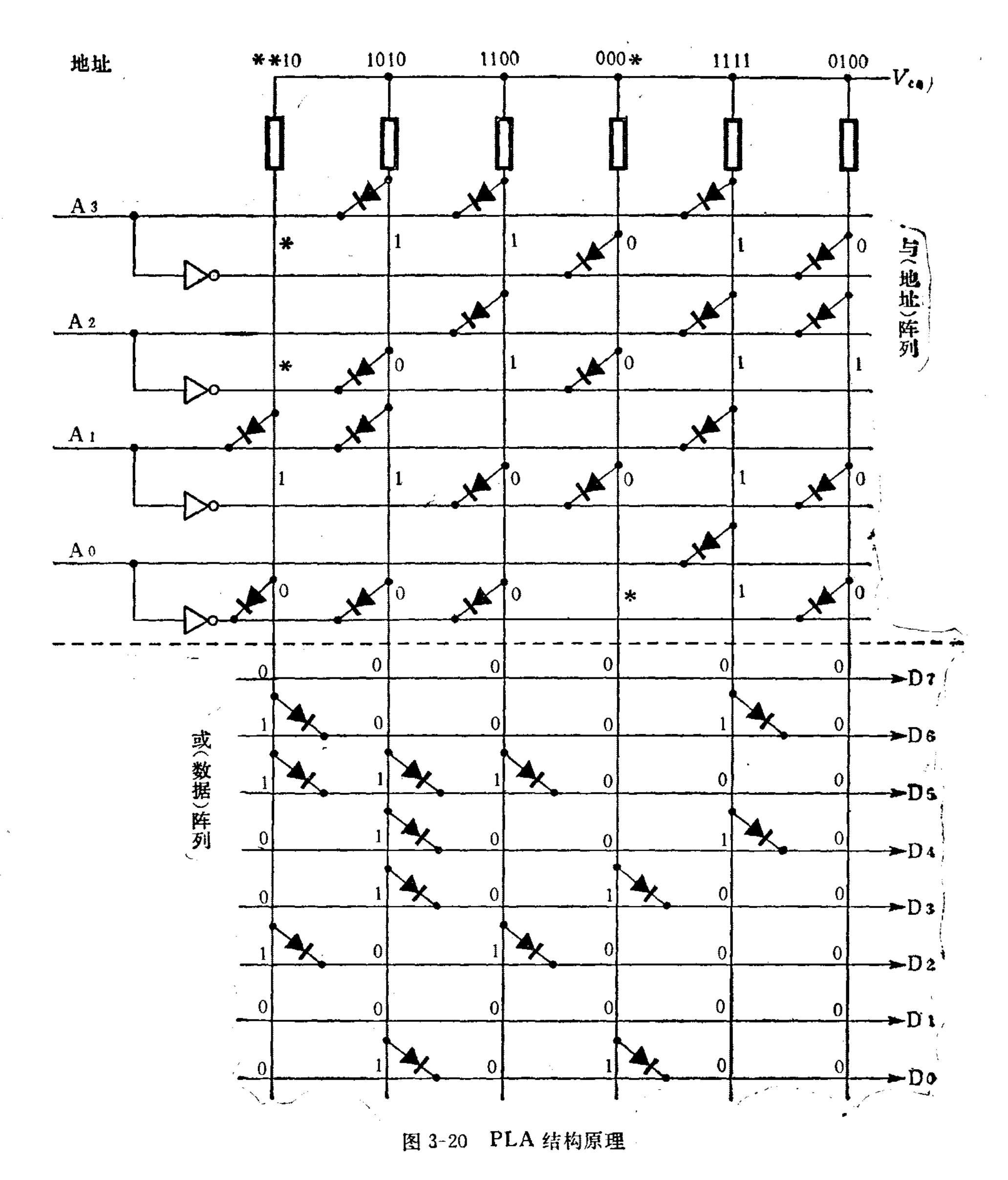


图 3-20 PLA 结构原理

阵列”。向地址阵列送入一个地址，就能选中相应的列线(字线)并将存贮在该列的各个数据并行输出。例如图中若输入地址 1100，则输出端数据为 00100100，与存贮一致。

PLA 的与阵列也可以编程。它不必像通用译码器那样把所有最小项函数涵盖无遗，而是可以根据实际使用的地址，通过掩模构成只对部分地址译码的电路。地址阵列的一些行的某些位，可与该位地址信号的正反相输入线都不连接，形成“不管位”。这些行就成为不完全译码行。不完全译码有时是应该避免的，但在这里却为微程序设计带来灵活性。一方面，多个地址可以选通数据矩阵的同一条字线，例如第一条字线地址为 * * 10（* 表示该位地址线不参加译码的“不管位”）。因此，用 0 010，0 110，1010，1110 都可以选中它。另一方面，用一个地址又可能选通多条字线，例如地址 1010，便能同时选中对应于 * * 10和 1010 的两条字线，数据矩阵的输出就是它们各个控制代码的逻辑“或”——也就是把两行的有效信号合并在一起，相

当于同时执行两条微指令。

用PLA代替ROM作为控制存贮器，仍然保持阵列整齐、可编程、易修改的特点，对于同样的宏指令系统，阵列规模小于ROM，而且使用比ROM灵活方便，因而得到广泛应用。Z80的微程序控制器就是PLA型。它以一个正逻辑的与非门阵列作为地址阵列，一个负逻辑的或非门阵列作为数据阵列。它们占据了芯片中央的主要部位。

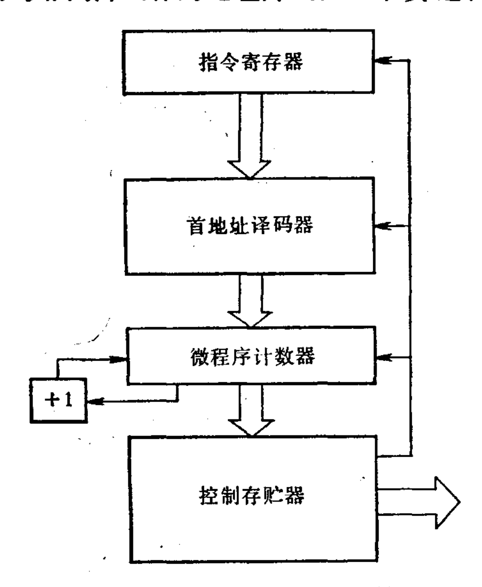


图 3-21 微程序计数器方式的控制器

（五） 微程序的执行逻辑

1. 微地址序列的产生。

如同宏指令的执行一样，执行微程序也必须连续提供一个微地址序列，将各条微指令的控制代码顺序取到控制线上。因此需要一个寻址机构。地址序列的产生，同时也是实现时序逻辑的手段。译码器是基本的寻址机构。但是，对宏指令译码只能给出一个微程序的首地址。后续地址的产生可采取不同的方式。

一种是以硬件为主的计数器方式(图3-20)。同宏指令计数器PC相似，用微程序计数器μPC来存贮当前执行的微指令地址，并按时钟脉冲自动增1。当微程序的结构也像宏指令组成机器语言程序那样按地址增序存放，就能循序执行。

微程序是分为“取指令”和“执行指令”两部分的。取指令微程序是一个固定的公共微程序(也称为“微例行程序”)。MPU的复位操作的重要内容就是将取指令微程序首址装入计数器，以便利用它取入第一条宏指令。显然，以0作为它的首址是很方便的，用一个计数器复位信号，便可指向取指令微程序了。取得的指令操作码译码后装入微指令计数器，就进入执行周期。在每一个执行指令微程序的最后一条微指令中，含有一个微程序计数器复位控制信号，配合时钟脉冲，就可以转入取下一条宏指令的阶段。

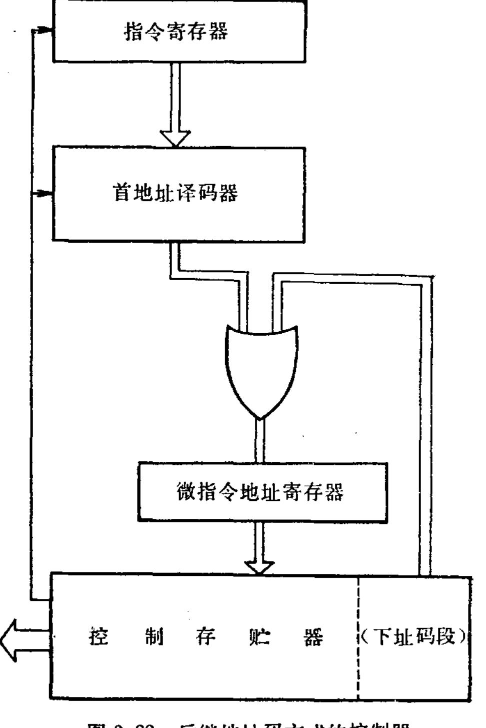


图 3-22 后继地址码方式的控制器

另一种寻址机构是以软件为主的地址链方式（图3-22)。它只需一个不增量的微地址寄存器。微指令中含有类似于BASIC语句“链指针”的地址码段，指出后续微指令的地址。地址码段的输出送入微地址寄存器，以供下个时

钟周期执行。执行指令微程序的最后一条微指令含有取指令微程序的首址，以实现指令周期循环。采用这种方式时，指令可根据需要任意排列，执行转移十分方便。但当微指令总数较多时，微地址码也相当长，需要多占存贮空间。

第三种是前面已介绍过的时标控制方式，用一个“时钟”产生机器周期和时钟周期的时标信号，与指令代码共同作为地址阵列的输入，即可使微程序按时序执行。

2．宏转移和微转移

微程序的执行也存在转移问题，由执行指令微程序转入取指令微程序便是一种常用的微转移。在微程序设计上，为了共用某些微指令，也需要使用微转移技巧。但是必须说明，微转移和执行宏转移指令是不同的两回事，执行无条件的宏转移指令时并不需要微转移。

现在先分析无条件转移和跳转类的宏指令的微程序。宏转移指令 JP NN 的微程序主要内容（取指令周期 M1 略）是：

M2　PC→ADR，PC＋1→PC，DBR→Z（将目标地址低字节取入中间寄存器Z）。

M3　PC→ADR，PC＋1→PC，DBR→W（将目标地址高字节取入中间寄存器W）。

这个微程序共 6 条微指令，至此结束，转入下一个取指令周期。但这个后续的取指令微程序前两条微指令的部分微操作和例行程序有所不同，不是　PC→ADR，PC＋1→PC，而是　WZ→ADR，WZ＋1→PC。

这两条微指令也是执行宏转移指令的公用微程序。上述 M3周期结束后转入，再由它转入取指微例行程序的第三条微指令。这样，就取入了 NN 地址的宏指令，实现了宏转移。

微转移对于含有地址码段的微程序系统是不成问题的。采用计数器方式时，也需要在微指令中给出目标地址。一般可采取“共用字段”的方法，以一个在微转移时不使用的微操作码字段作为临时的地址码段。为了区别，这个字段应增加一个标识位，如标识为 0，表示是微操作码；标识为 1，表示是转移地址，通过标识码控制多路转换器将它送入微程序计数器，实现微转移。

执行条件转移和跳转的宏指令时，微程序也需要有分支。以宏指令 JP cc，NN 的微程序为例。在M1取入指令后，立即对条件标志进行测试。测试电路原理见图 3-11。在宏指令所含条件码的控制下，电路送出负逻辑的测试结果—— 0 表示条件为真，1 表示条件为假。

用测试结果控制微程序分支，可有多种方式。采用计数器的，在增量的基础上再加测试真值，就会产生两个相差 1 的地址。其中原来的后续地址是“真”分支，应是一条转移微指令，转入目标微程序入口。下一个微地址是“假”分支入口，按常规顺序执行（见图3-23）。采用地址码段的微程序，微地址寄存器中应保留一个“条件位”，由测试真值置位，从而实现程序分支。

微程序控制器还可设有内部堆栈，用来保存当前微地址，以进行微子程序调用。

3．微指令的重叠执行。

每一条微指令所包含的微操作都只涉及 MPU 的局部。对于 PLA 控制器，如果两条微指令的微操作在数据通道和时序方面互不矛盾，就可以叠加在一起执行。

在讨论周期和时序时，已经看到它们的主要操作是访问内存或外设，而运算只需“镶嵌”在访外的周期中进行。例如，在 Z80 指令表上，可以查到 ADD A，r（r 表示一个通用寄存器）这条指令只需要 4 个时钟周期，即用取指周期就能完成。M1周期在 T3 上升沿取入指令，T3 周期用来对指令译码。而 T4 的一个周期要完成将两个操作数送入 ALU 运算并送回累加器

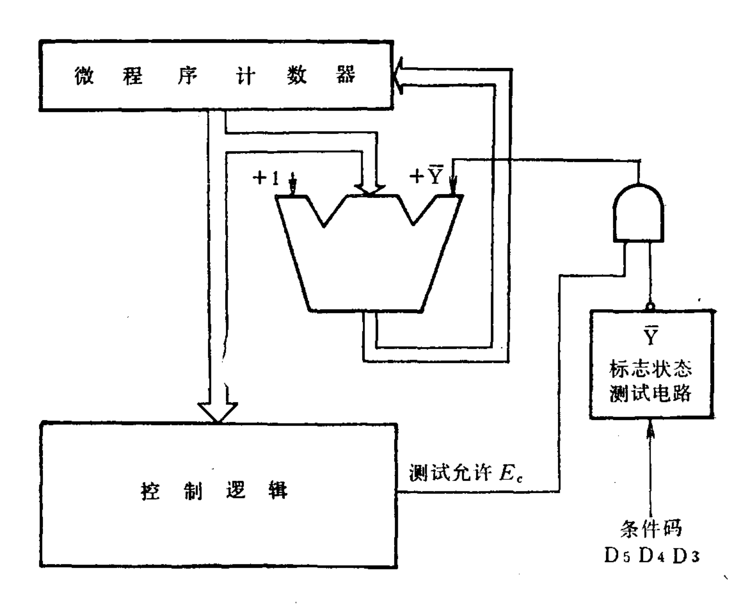


图 3-23 条件微转移方式

是不够的，势必需要延长一个T5周期。这个T5周期也就结束了本指令周期，随后应是下一个M1周期。M1周期的T1是发送地址和系统控制信号，并不涉及内部数据总线、累加器和运算器。所以，把前面的T5和后面的T1重叠操作是可行的。于是有

T4　　A→ACT，r→TMP

T1　　(ACT)+(TMP)→A

　　　PC→ADR……

可见，这条指令实际用了5个T周期，但只占4个T周期的时间。这种技巧在算术逻辑运算指令周期中是普遍使用的。必要时，后续取指周期的T2也可以同样利用。

这一方面，8080和Z80有一定的差异。8080的M1周期的T1，需要使用数据总线输出状态信息，因而T1不能供重叠使用，只能利用T2完成前一指令遗留的运算操作。

4．微程序与中断。

采用微程序控制器的MPU，对外部中断信号也以微程序方式响应，使硬件得以简化。

MPU以中断触发器接受由引脚送入的中断信号，中断允许触发器IFF用作输入控制。执行开中断指令后，IFF置位，允许中断信号进入。在微程序执行的一定阶段，如Z80是在每个机器周期的最后一个时钟周期，微指令中含有中断检测信号。若检测到中断请求，中断逻辑便会提供一个固定的微地址，使执行转入中断响应周期的专用微程序。每一种类型的中断（如Z80的可屏蔽中断、不可屏蔽中断和DMA请求），分别用一个专门的微程序处理。一般先要将程序计数器PC中的断点地址压入堆栈保存，再按规定途径取得中断服务子程序地址，转入执行，进入常规循环过程。

第四章　硬系统剖析

本章将通过对标本机机器系统——即通常所称“硬件”结构的剖析，具体讨论用各种逻辑器件怎样组织成一台可供实用的计算机。

LASER　310 作为一种实用机型，结构相当简洁，是一个较好的标本。它基本由通用集成电路组成系统，脉络清晰，易于理解。机型发展后期采用了三片门阵列 IC 制成的 GA 系列专用电路，但原有逻辑基本未变。为了帮助读者理解它们的工作原理和便于维修代换，我们分别给出了 GA 芯片的“解析图”，以通用 IC 的组合来表示其结构。当然，这里主要着眼于基本逻辑关系，具体电路细节不一定与实际相符，仅供参考。

图 4-1 是全机总体逻辑图(附书末)。

一、CPU

LASER　310 的 CPU 采用 Z80A 微处理器。MPU 用作计算机系统的中央控制单元，应配置必要的支持电路，并根据系统的规模、功能要求和总体结构，恰当地将各个引脚接入电路。

(一)　Z80A 的引脚使用

数据总线：LASER　310是八位机，8 条数据总线全部使用。

地址总线：由于LASER　310基本系统内存最高地址为 B7FFH(并可扩充到 FFFFH)，必须使用全部 16 条地址总线。

控制总线：

1. 基本系统使用$\overline{RESET}$、$\overline{MREQ}$、$\overline{IORQ}$、$\overline{RD}$、$\overline{WR}$、$\overline{INT}$和$\overline{RFSH}$。
2. 基本系统不使用$\overline{M1}$和$\overline{HALT}$两个输出信号，但将它们接到扩展插口以供外设使用。
3. 基本系统不使用$\overline{WAIT}$和$\overline{NMI}$ 两个输入信号，除将它们接到扩展插口外，因为 MOS 电路不允许输入端悬空，故同时通过电阻接 + 5 V，使其通常保持无效状态。当扩展插口输入低电平信号时，由于电阻上产生压降，CPU 仍能得到低电平有效信号。
4. 总线控制信号$\overline{BUSRQ}$和$\overline{BUSAK}$在系统中弃之不用，输入的$\overline{BUSRQ}$ 接 + 5 V，输出的$\overline{BUSAK}$ 悬空。因此，用户若需使用它们必须另行接线。

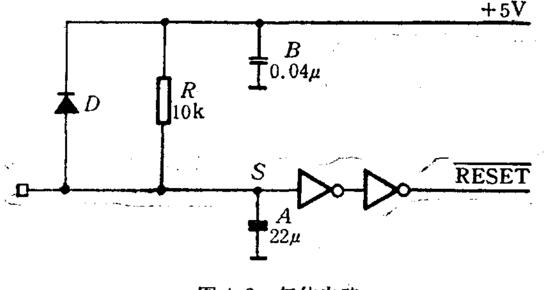


图 4-2　复位电路

(二)　CPU复位电路

CPU 每次开始工作时，需要外加一个复位信号以进入初始状态。Z80 的复位信号是宽度 ≥ 3 个T 周期的 TTL 电平负脉冲，MPU 复位后此信号即应变高无效，以免影响工作。LASER　310　的复位电路见图 4-2。在打开

电源开关的瞬间，电流通过电阻 R 向电容 A 充电，电阻上产生压降，使 S 点成为低电平。此信号经 74LS04 中两个非门串接的缓冲器送给 Z80A 的$\overline{\text{RESET}}$ 引脚。阻容网络的时间常数能够满足 Z80 复位的要求。电容充电结束，S点电平升至接近 + 5 V，从而$\overline{\text{RESET}}$ 信号无效，MPU 正常工作。若电源断电，电容A中的电荷通过二极管及小电容B放电，迅速作好下一次复位的准备。

LASER 310 未设 RESET 键，复位电路只在开机时起作用。当程序陷入死循环，BREAK键无效时，只有关机重开，有时会造成很大损失。由于复位电路的一个输入端通往扩展插口，故可由外设用低电平信号使 MPU 复位。一个最简单的办法，是当机器“锁死”时，用小改锥把内存扩展插口的$\overline{\text{RESET}}$ 引脚与相邻的地线脚短接一下，MPU即可复位。这时用户程序区内容基本未变(见第六章)，很容易恢复。如在两脚间接一常开按钮开关，便成简单的RESET键。

(三) 时钟电路

LASER 310系统时钟电路见图 4-3。它用一只固有频率 17.7MHz 的石英晶体为核心的振荡器作脉冲发生器。选择这一频率是为了兼顾 Z80A 和视频系统的需要，使系统主频经整数分频后能获得 PAL 制的各种定时信号，同时又符合 Z80A 对时钟频率(≤4MHz)的要求。

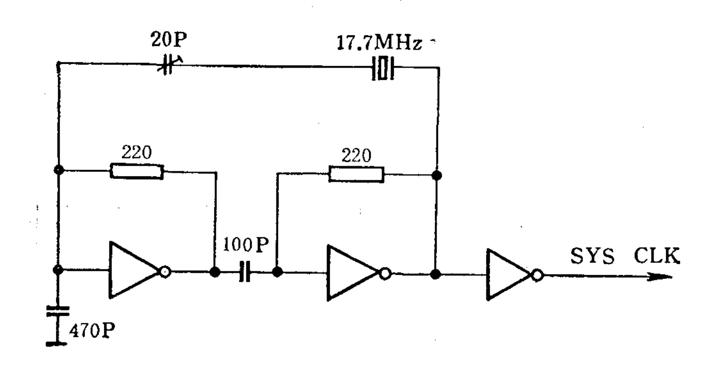


图 4-3 系统时钟电路

GA008 中有两个分频器，分别将系统时钟信号四分频为视频系统基本的时钟信号 PAL CLK(即彩色副载频 4.43MHz)，五分频为 Z80A 的时钟信号 ϕ(3.54MHz)。

图 4-4 是 GA008 中分频器部分的解析图。本图的四分频器用双 D 触发器(74LS74)接成的四进计数器。因为$\overline{\text{Q}}$端反馈到 D 端同时作为下一级时钟信号，CLK输入四个脉冲，Q2输出一个脉冲，使频率变为原来的 1/4。五分频器相当于十进制计数器 74LS90 的五进制部分。

二、内 存 贮 器

存贮器是计算机内部的工作空间，用以存放系统程序、应用程序、操作数据和运算结果。一般说来，存贮器容量越大，计算机功能越强。学习机的内存容量一般都比较小。

微型计算机的内存贮器按用途可分为驻机程序存贮器、主存贮器和缓冲存贮器等种类。LASER 310 的系统程序存贮于一片掩模 ROM(HN613128)中，主存贮器是 8 片 DRAM

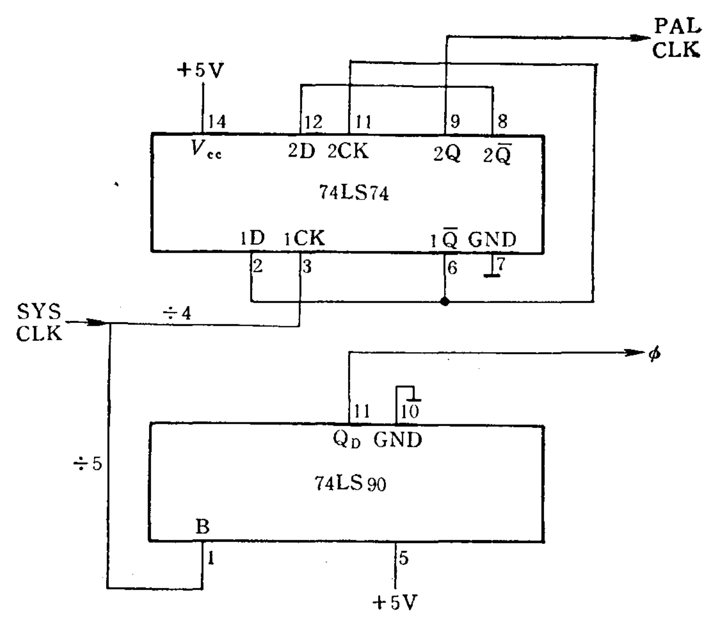


图 4-4 GA008的系统时钟分频器

(4116)。显示缓冲存贮器是一片 SRAM(6116)。显示存贮器(VRAM)将在“视频电路”一节介绍。

（一） 驻机系统程序存贮器

系统程序的载体——只读存贮器 613128，是一片 128Kbit 容量的掩模 ROM。因为 Z80 初始化后由地址 0000H 开始执行第一条指令，为使其自动进入系统程序，ROM 起始地址取 0000H。本机系统程序 V2.0 的规模为 16KB，结束地址为3FFFH。这一空间对应于A15A14 = 00的所有地址。在系统中，ROM的片选信号$\overline{CS}$由主译码器输出的$\overline{R0}$和$\overline{R1}$经负逻辑的二极管或门合成。其中任一个有效，ROM 即被选通。$\overline{R0}$由 A15A14A13 = 000 译码产生，$\overline{R1}$由 A15A14A13 = 001译码产生。这是早期 200 型使用两片 ROM 所留下的遗迹。

ROM 是只读不写的，控制比较简单。芯片选择信号$\overline{CS}$无效时，数据输出端呈高阻态。在$\overline{CS}$有效的条件下，直接以 Z80A 的 $\overline{RD}$ 信号作为输出允许信号$\overline{OE}$，控制将数据取入总线。

（二） 主存贮器

1. DRAM 4116。

4116 是动态 RAM，存贮单元结构见第二章。单片容量 16Kbit，数据 I/O线各一条，每次只能读写一位数据。为了以字节为单位存贮数据，采取“位片”结构，以 8 片各连接一条数据总线，同时并行读写。八片总容量为 16K BYT。

图 4-5 是 4116 内部逻辑图。存贮单元排列成 128 行 × 128 列的矩阵，每一列的正中，即第63列和第64列之间有一个“读出再生放大器”。存贮矩阵用 7 位行地址和 7 位列地址信号重合选址，经行译码器和列译码器译码后使指定的行线、列线成为高电平，在内部各个时钟信号的控制下，向选中的存贮单元读写数据。

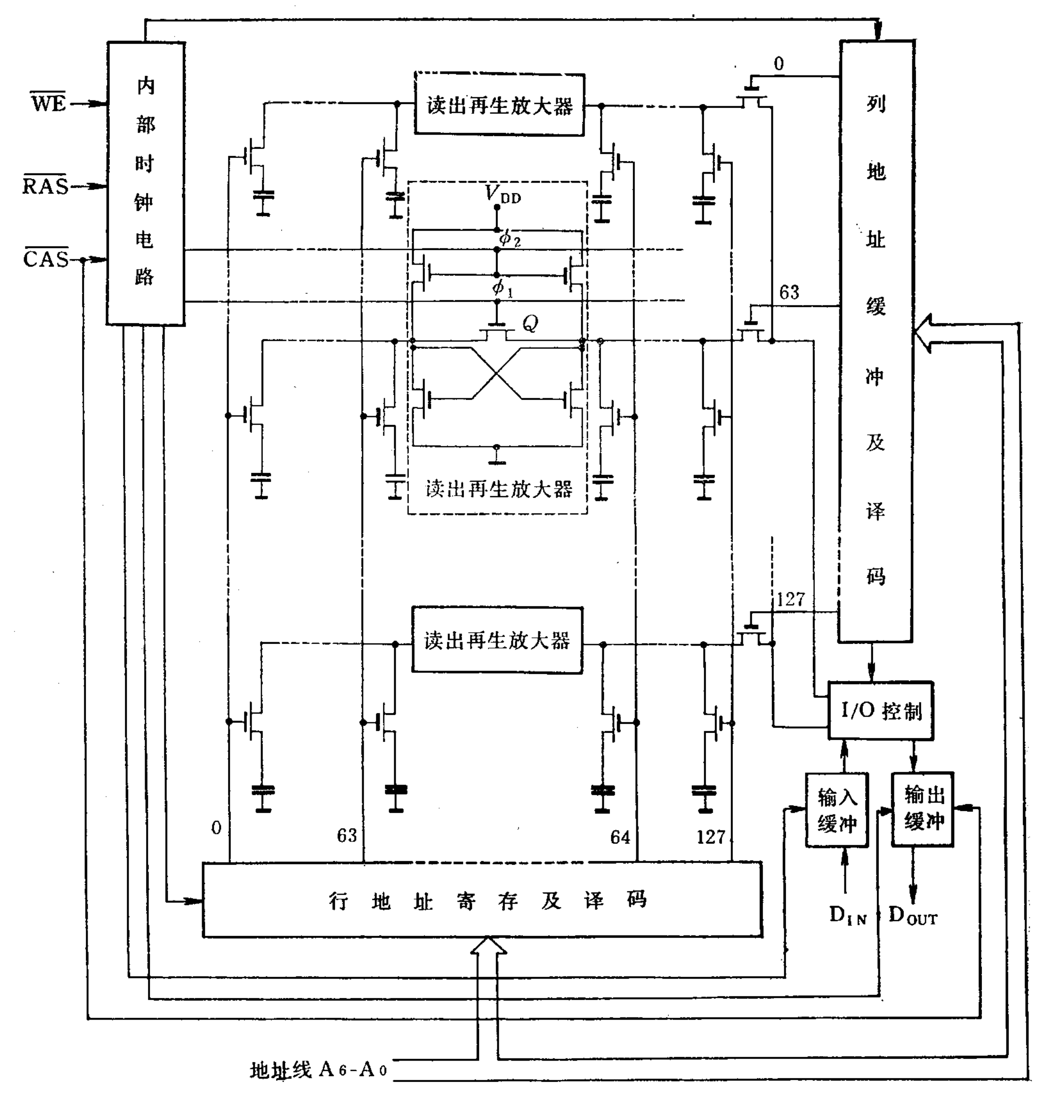


图 4-5 4116逻辑图

为了减少引脚数和降低封装成本，4116采取地址线复用方式。行、列地址信号通过同一组地址线分时送入，在行地址选通信号$\overline{RAS}$和列地址选通信号$\overline{CAS}$的控制下，先后存入内部的行、列地址缓存器，然后译码。在LASER 310中，主存地址译码、行列地址转换和列选通信号$\overline{CAS}$的产生，均由 GA008 的担负。而以 CPU 的$\overline{MREQ}$信号直接作为行地址选通信号$\overline{RAS}$。

读出放大器是一个时钟控制的双稳态触发器。在读操作时，最初时钟 $\phi 2$ 为低电平，触发器因“关断”了电源而不工作。先令 $\phi 1$ 电平变高，使 Q 导通，读出放大器两端电位平衡，成为 0 电平和 1 电平之间的状态，然后 $\phi 1$ 降低，Q关断，两端被隔离起来。行选择的高电平信号将一行的 128 个存贮单元控制管全部打开，各列读出放大器的一端同选中行的存贮电容接通，此端的电平将因存贮电荷的影响升高（存有 1 时）或降低（存有 0 时），与读出放大器另一端间产生了电位差。此时，$\phi 2$ 变高，使触发器进入工作状态，由于输出与输入相互的正反馈，两端电

位差愈来愈大，直至达到1电平和0电平的最大幅度为止，并保持稳态。连接选通行单元的一端信号电平与原存贮信息一致，相当于存贮信息的放大。因为这时控制管并未关闭，放大器的稳态电平又反馈给存贮电容，恢复了在控制管打开后电荷流动所破坏了的原存贮信息。这时，当某一列线被选中成为高电平，128个已准备好数据的读出放大器之一便可将数据信号经列选择管送出。因为Q已关断，只从触发器一端读数，若是另一侧各行的数据，则正好相反，须再经一次反相后输出。最后行选择信号无效，各控制管关断，存贮数据被保存起来。

在上述读操作中，虽然只从一个读出放大器读出数据，但同一行的128个放大器都在工作，准备好了数据，实际上也就同时实现了对全行存贮单元的充电刷新。可见每读数一次就有128个单元得到刷新。

写入数据的过程比较简单。行、列地址选通后，$\overline{WR}$信号有效，数据便可送入存贮电容。

由上述可见，DRAM的工作需要相当复杂的定时信号，由芯片的时钟电路在$\overline{RAS}$、$\overline{CAS}$和$\overline{WE}$三个外部定时信号的基础上生成。DRAM与CPU配合时，对定时的要求较高。特别是在Z80取指周期，指令需在时钟脉冲T3的上升沿取入CPU，从地址信号发出，行列地址转换、译码，到数据取到总线上的繁多步骤，必须在两个时钟周期内按严格的时序完成。行、列地址应分别超前于$\overline{RAS}$和$\overline{CAS}$信号出现在DRAM地址线上，$\overline{RAS}$和$\overline{CAS}$之间应有足够的时间间隔。

因为DRAM的存贮电容很小，会很快放电使数据1丢失。一般情况下数据保持时间虽可接近1秒，但为确保数据的安全，规定必须在2ms之内就对它充电刷新一次。读操作虽能刷新一行单元，但读的次数和行数都是随机的。因而Z80在取指周期的后部专门对DRAM轮翻刷新。即用行地址和$\overline{MREQ}$信号选中一条行线，利用读出放大器对存贮电容的充电过程实现刷新。但未给列选择信号，数据并不取出。

显然，这种在取指周期刷新的方式，每次的间隔随指令周期的长度而不同。最短的只间隔4个T周期(执行连续两个取指周期的双操作码指令时)。Z80最长的指令周期有23个T周期，两次刷新间隔可达19个T周期，即使连续128次执行这一条指令，钟频4MHz的Z80A刷新内存一次的时间为$0.00025ms \times 19 \times 128 = 0.608ms$，远小于2ms。这样的极端情况，事实上不可能出现，所以钟频1MHz的Z80也能满足≤2ms的DRAM刷新周期要求。

如果系统使用$\overline{BUSRQ}$或历时过长的$\overline{WAIT}$控制信号，使得Z80不能及时对DRAM刷新时，应另设补充性的刷新电路。LASER 310基本系统没有使用$\overline{WAIT}$和$\overline{BUSRQ}$，刷新操作不受影响。

4116需用三种电源：$V_{CC} = +5V$，$V_{SS} = -5V$，$V_{DD} = +12V$，由系统的电源变换器供给。后来研制的大容量DRAM已改进为单电源(+5V)供电。

2. 地址多路转换器。

地址多路转换器用来将指向主存贮器的地址信号分解为行、列地址信号，分时发送给DRAM。它是GA008的一个部分。图4-6是我们给出的16KB模式解析图，由两个二选一多路选择器(74LS157)构成。地址总线A7～A0和A13～A8分别接到A、B两组输入端。输出端MA6～MA0分别连接各片4116地址线；MA7在16K DRAM系统中不用。若设置为16K/64K可选模式，16条地址总线应通过多路分配器接入不同的引脚。

多路选择器控制端$\overline{G}$接地，使其保持在工作状态。选择信号S为低电平时输出A组(行)

地址，为高电平时输出B组(列)地址。

三、译码器

(一) 系统主译码器

早期的LASER 200型，系统较小，用一个组合译码器即可管理全部内存和基本I/O接口。后来的310型增加了16K DRAM，并配置了专门的译码器，系统主译码器只管理主存贮器以外的、内存空间低端的各个器件。图4-7是主译码器解析图，由双2-4译码器(74LS139)组成。$\overline{MREQ}$为译码允许信号；$\overline{MREQ}$无效，各输出脚均为高电平无效，各目标器件都被关闭。$\overline{MREQ}$有效时则根据地址总线A15～A11的信号产生相应的片选信号，使目标器件选通。每个信号对应的地址范围，是译码位之外各位全为0到全为1之间的所有地址，见图4-8。

从解析图上可以看到，译码器还有两个输出信号未用，因为系统将4000H～67FFH的空间保留来扩充DOS，然而后来配置的磁盘驱动器接口又自带有译码器。这两个信号可在扩充系统功能时加以利用。

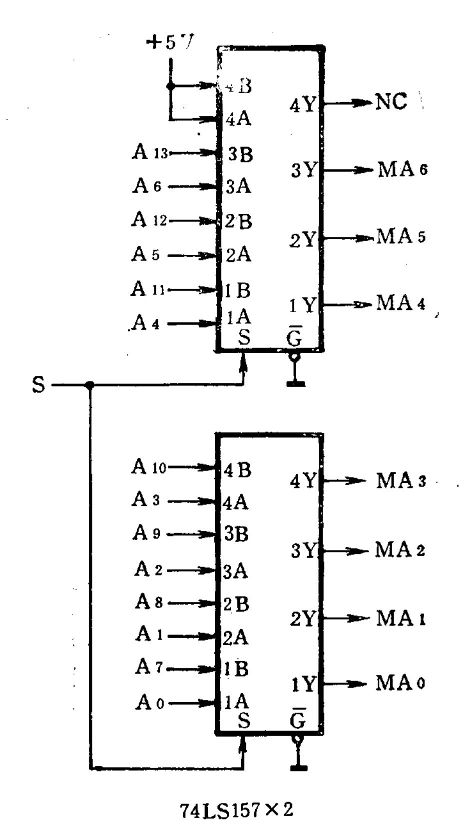


图4-6 主存贮器地址多路转换器（16K模式）

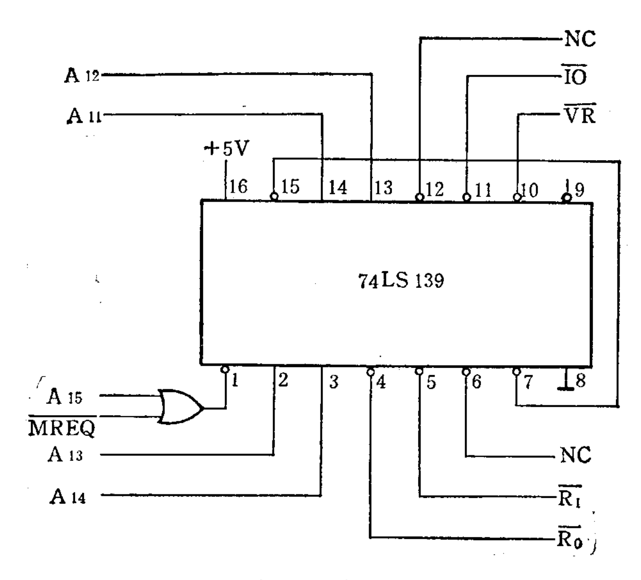


图4-7 主译码器

A15	A14	A13	A12	A11	$\overline{R0}$	$\overline{R1}$	$\overline{I/O}$	$\overline{VR}$	目标器件	对应的地址范围
0	0	0	*	*	0	1	1	1	ROM 0	0000H～1FFFH
0	0	1	*	*	1	0	1	1	ROM 1	2000H～3FFFH
0	1	0	*	*	1	1	1	1	未用	4000H～5FFFH
0	1	1	0	0	1	1	1	1	未用	6000H～67FFH
0	1	1	0	1	1	1	0	1	基本I/O接口	6800H～6FFFH
0	1	1	1	0	1	1	1	0	VRAM	7000H～77FFH

图 4-8

（二） 主存贮器地址译码(控制)器

系统为低端各器件分配地址为 0000H～77FFH，因而主存贮器地址从 7800H 开始。GA008 的译码器直接对CPU地址总线信号译码。基本系统配置的16KB DRAM地址的地址范围 7800H～B7FFH，地址信号高 5 位应为 01111～10110。这个译码器还可以用于 64K 存贮器(如4164×8)，用 GA008 第 7 脚的“$16/\overline{64}$”信号选择，高电平时取 16K 模式，低电平时取 64K 模式。64K 模式的地址范围 0000H～FFFFH，但低端部分空间与系统的内存分配是冲突的，需作特殊处理。LASER 310 基本系统将这个引脚固定接 +5V，只按 16K 模式工作。

译码器实际上是一个产生地址多路分配器的切换信号S和列选通信号$\overline{CAS}$ 的控制电路，以向DRAM送入列地址。解析图见图 4-9。译码器对$\overline{MREQ}$和高五位地址信号A15～A11译码，并受$16/\overline{64}$选择电平控制。

逻辑条件符合时，S输出高电平，否则输出低电平。

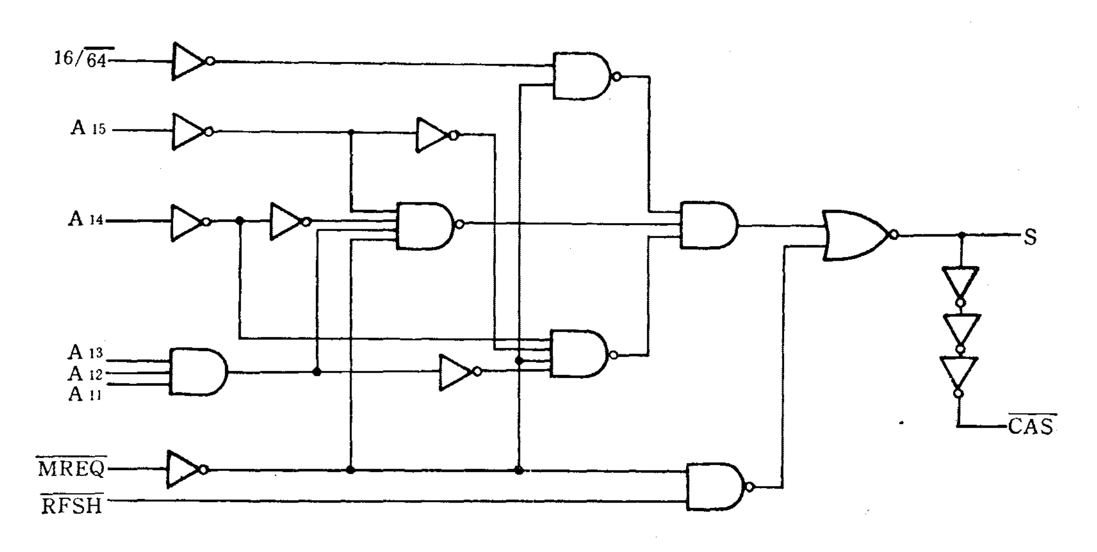


图 4-9 主存贮器地址译码器

S 经三个非门延时并反相，成为$\overline{CAS}$。

在机器周期的T1，CPU发出内存地址信号，但$\overline{MREQ}$尚无效，译码器输出 S 为低电平，$\overline{CAS}$无效，多路分配器将 A6～A0 送入 DRAM 地址线。T1下降沿后，$\overline{MREQ}$变低，成为 DRAM 的$\overline{RAS}$信号，将地址打入行地址缓存器。此时，若取 64K 模式，S 变为高电平；若是 16K 模式，则在高五位地址符合 01111～10110 范围时 S 变高，多路器将 A13～A7 送给 DRAM 地址线。S 经延时使$\overline{CAS}$变低，将地址打入列地址缓存器。到此就完成了译码器的使命。

在 M1 周期的 T3，Z80A将刷新地址送上地址总线，S 为低电平，多路器将 A6～A0 送 DRAM，$\overline{MREQ}$变低后打入行地址缓存器。这时由于$\overline{RFSH}$是低电平，使S不能变高，$\overline{CAS}$维持无效。所以只起到启动 128 个读出再生放大器对一行存贮单元充电刷新的作用，并不读写。

四、I/O设备和接口

计算机通过输入/输出(I/O)设备和外界交换信息。常用的 I/O 设备有键盘、显示器、打印机、磁盘驱动器、磁带录音机和扬声器等等。它们的数据通道应同主机的数据通道相通，但又不能简单地硬接在一起，而必须经过一定的接口电路。接口的主要作用一是开关控制，用地址和有关控制信号选通指定设备的数据通道；二是时序协调，例如将输出输入的数据锁存在接口中，等待对方读取；三是缓冲驱动，用三态或双态驱动门以增强驱动外部逻辑器件的能力，以及用于总线连接。

计算机对外部设备的寻址有两种方式，一种如 Z80 系统的专用 I/O 口方式，每个外设可由一个8位的I/O口地址和$\overline{IORQ}$信号选通，进行读写。另一种是内存地址方式，将外设当作内存一样地编址和读写。LASER 310 两种方式兼而用之。键盘、蜂鸣器、磁带机和显示器等基本 I/O 设备采取内存地址方式，主板上配置有接口电路。打印机、磁盘驱动器等作为扩充外设，以接口部件相连接，用 I/O 口地址寻址。

(一) 基本 I/O 设备

1. 键盘。

LASER 310 键盘是非编码式的，本身不能产生字符代码。每个键是一个常开开关。各个开关的一侧端头分别用行线连成 8 行，另一侧端头用列线连成 6 列。行线作为地址线，分别连接地址总线A5～A0，列线作为数据线，连接 GA004 的键盘输入接口。45 个键排成 6 × 8 的矩阵(三个位置未用)，如图 4-10 所示。

这种键盘之所以能够用来输入字符，主要靠系统软件中键盘检测和译码程序的作用。键盘矩阵的六条列线是通过接口连接数据总线D5～D0的，列线输出数据可由 CPU 读取。由于键盘列线的另一端通过电阻接＋5V，所以它们通常保持高电平，无按键动作时 CPU 将读得数据 * * 111111B。如果使某一行线成为低电平，而这一行中有某一键被按下，行线与列线短路，这一列线输出电平必将降得与此行线一致。这时CPU 读得的键盘数据，与该列相应的位就为 0，其余位仍为 1。既然预先已知输出低电平的行，又从数据中获知读出 0 的列，便可以判断是位于该行该列交点的键被按下了。

键盘检测的方法，就是循环地向每一行线发送低电平信号，也就是用该地址线为“0”的地

	KD5	KD4	KD3	KD2	KD1	KD0	扫描用地址
A 0	R	Q	E		W	T	68FEH
A 1	F	A	D	CTRL	S	G	68FDH
A 2	V	Z	C	SHIFT	X	B	68FBH
A 3	4	1	3		2	5	68F7H
A 4	M	空格	,		.	N	68EFH
A 5	7	0	8	-	9	6	68DFH
A 6	U	P	I	RETURN	0	Y	68BFH
A 7	J	;	K	:	L	H	687FH

图 4-10

址去读取数据。例如，检测第一行时，使 A0为 0，其余为 1，加上选通键盘接口的高五位地址01101,成为01101 * * * 11111110B(A8～A10可为任意值,这称为"不完全译码",所以68FEH,69FEH,6AFEH,6BFEH,6CFEH,6DFEH,6EFEH,6FFEH 均等效)。如此时这一行的"E"键按下,则相应列 KD3 输入的信号为 0，其余列仍为 1，读得数据 * * 110111B,由此便可判断是第一行第四列的 E 键。如这一行未按键(各位全为 1),则将地址中的 0 左移一位(01101 * * * 11111101B,即 68FDH,……,6FFDH),检测下一行。六行检测完毕无按键再周而复始,在机器语言级高速地循环扫描,人手击键虽快,也是不会漏过的。

确定是某一键后,监控程序便可通过计算、查表等取得相应字符代码,存入显示 RAM 并显示在屏幕上,看起来就像是键盘本身输入了字符一样。

2. 数据磁带机。

计算机的磁带记录信号都是逐位顺序传送的串行信号,以bit为传送单位。LASER 310 的磁带记录信号是非标准格式,信号波形见图 4-11。

LASER 310 的记带信号,未使用专门的电路,完全用软件方式生成。将待写入磁带的字节分解成位,再按各位是 1 或是 0,分别用一个子程序通过记带输出接口发出上述脉冲波形,送到磁带机 MIC 插口。

磁带机输入的信号,经放大和整形后(见图 4-12),由输入接口送入数据总线的D6。CPU 用软件方式逐位读取,再"装配"成为字节。

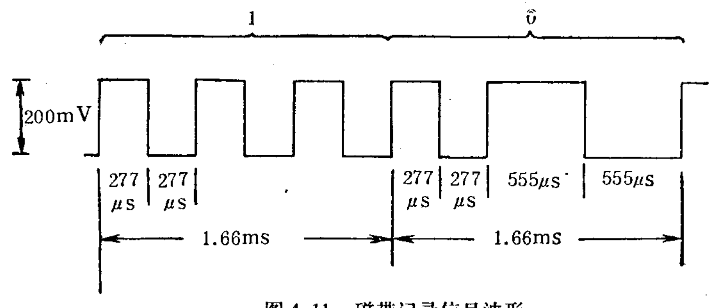


图 4-11 磁带记录信号波形

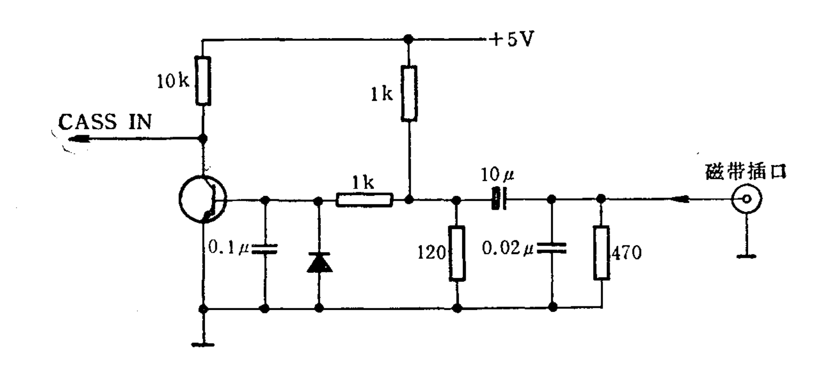


图 4-12　磁带信号输入电路

3．蜂鸣器。

LASER　310用压电陶瓷蜂鸣片作发声设备。它的阻抗很高、耗电极小，但音量也小，音质较差。发声的方法是使它的两端保持电位差，并按一定频率改变电场方向——即使两端电平高低互换。由于压电效应的驱动，蜂鸣片振动发声。声音频率由电场变化频率决定。

蜂鸣片两端的电平信号，由CPU向接口连续写入不断变反的数据来获得。

（二）　基本 I/O 接口电路

基本 I/O 设备的接口中，显示器接口电路比较复杂，这就是在系统中占有很大比重的"视频电路"。其余设备的接口（以及几个视频控制口）已集成在一片门阵列 IC　GA004 中。

图 4 -13 是GA004 解析图，可由四片通用集成电路组成，它们是：八总线驱动器（74LS244），八总线接收/驱动器（74LS245），八 D 触发器（74LS273），四或门（74LS32）。

1．键盘及磁带机输入接口。

键盘有 6 条列数据线KD5～KD0，它们通过八总线驱动器的 6 个三态门接到数据总线。另用一个三态门作为磁带信号输入（KD6）接口。KD7 用途比较特殊，被用来"监视"视频显示发生器 MC6847 的垂直同步信号$\overline{\text{FS}}$。它在场消隐期变低，被系统作为对 CPU 的$\overline{\text{INT}}$请求信号，用来启动监控程序，以便输出显示，是一个很重要的信息。

这个八总线驱动器的允许信号$\overline{\text{E}}$，由 CPU 的读内存信号$\overline{\text{RD}}$和系统译码器输出的片选信号$\overline{\text{IO}}$经负逻辑与门合成。也就是说，当CPU发出6800H～6FFFH 中的任一地址和$\overline{\text{RD}}$信号，$\overline{\text{E}}$即变得有效。这时，以上 KD7～KD0 的三组输入信息就进入数据总线，送给CPU。不符合上述条件时$\overline{\text{E}}$无效，各三态门均呈高阻态，使各输入设备同数据总线隔离。

2．显示 RAM 接口。

作为显示 RAM 的 6116，本身带有三态输入/输出驱动器，可与 Z80 数据总线直接连接，并不需要另设缓冲接口。但在系统中它的数据线同时连接着视频显示发生器，并要不断地向视频系统输出显示信息。所以将它直接挂在数据总线上无异于将 Z80 和 6847 的数据线短接。这是不允许的，故用一组双向三态门进行隔离。八总线接收/驱动器由两组各八个方向相反的三态门组成。数据总线 D7～D0 接 *B* 组引脚，6116 数据线 D7′～D0′接 *A* 组引脚。接口由选通信号$\overline{\text{G}}$和方向选择信号DIR控制。$\overline{\text{G}}$为高电平无效时两组三态门都呈高阻态，CPU 数据总线和 VRAM 完全隔离，可供 VDG 访问。$\overline{\text{G}}$有效时，若 DIR 为高电平，*A* 组门开启，数据由 *A*



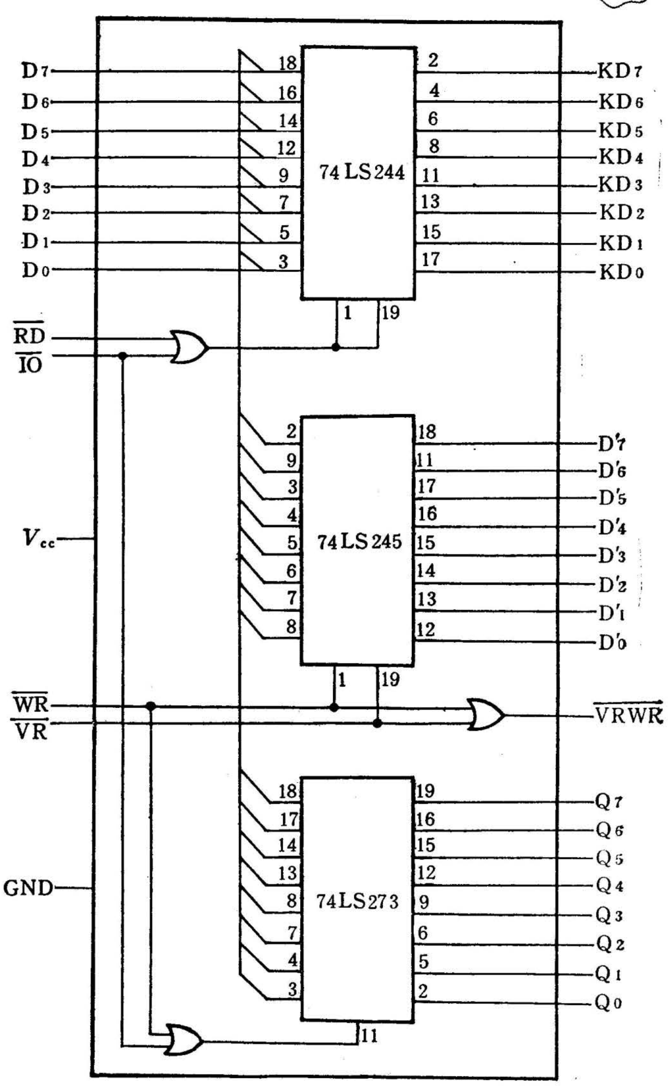


图 4-13 GA004解析图

方送到B方；若 DIR 为低电平，则数据由 B 方送到 A 方。在本系统中，$\overline{G}$ 用显示 RAM 的选择信号 $\overline{VR}$，DIR用CPU 的$\overline{WR}$控制信号充当。当 CPU 发出 7000H～77FFH 范围内的显示地址，则$\overline{G}$变为有效。若 $\overline{WR}$高电平无效，数据由 6116 读入数据总线，$\overline{WR}$ 有效时，CPU 的数据写入 6116。因为$\overline{VR}$信号同时作为6847的存贮器选择信号$\overline{MS}$，$\overline{VR}$有效即 $\overline{MS}$ 有效，将阻止 6847 发出访存地址，以避免冲突。

同时，GA004 中还用一个负逻辑与门将 $\overline{WR}$ 和 $\overline{VR}$ 合并为 $\overline{VRWR}$，作为 6116 的写选择信号。$\overline{VRWR}$ = 1 读出数据，$\overline{VRWR}$ = 0 写入数据。

3. 输出锁存器。

LASER 310 用一组八 D 触发器组成八位数据锁存器，锁存输出信息或控制信号，供基

本外设取用。每个触发器的 D 端分别接一条数据总线，Q 端输出数据。触发器由外部时钟脉冲CK下降沿触发。CK 由$\overline{IO}$与$\overline{WR}$经负逻辑与门合成。当 CPU 发出 6800H～6FFFH 间的地址和写内存信号，$\overline{IO}$和$\overline{WR}$都变低时，CK 负跳变，Q 端电平变得与 D 端相同，并能继续保持到时钟信号下一次负跳变——即执行下一条写基本输出接口指令时。

八个输出端分别用作——

Q7　　未用

Q6　　未用

Q5　　蜂鸣器 *B* 端电平

Q4　　CSS（屏幕显示背景色选择信号）

Q3　　$\overline{A}$/G（屏幕显示模式选择信号）

Q2　　CASS OUT（磁带记录信号）

Q1　　未用

Q0　　蜂鸣器 *A* 端电平

由于触发器 *Q* 端信息不能从 *D* 端取回，所以锁存器是“唯写”器件。

以上总线驱动器和 D 触发器在地址使用方面有以下特点：

第一，两者地址虽然相同，但因为还必须加上$\overline{RD}$或$\overline{WR}$合成选通信号，所以不会混淆。写指令指向锁存器，读指令指向总线驱动器。

第二，芯片并不具有二级译码器，属于不完全译码。除键盘扫描地址略有不同外，地址的低11位无意义，6800H～6FFFH中任一地址都是等效的。

（三）　扩充插口

LASER　310 主板后方备有两个扩充插口。它们的各个引脚，实际上就是板上各条总线的游离端。左边较窄的一个为 30 脚，其中地址总线只有A7～A0八条，只能用来扩充 I/O 设备，如打印机、游戏棒和光笔等。右边较宽的44脚插口，包含了除$\overline{BUSRQ}$和$\overline{BUSAK}$之外的 Z80 全部总线，并提供 + 9 V 电源，所以除可用于扩充 I/O 设备，还能用来扩充内存。扩充RAM卡和软盘驱动卡（含有ROM）都必须插接在这个口内。所以通常称为内存扩充插口。

I/O扩充插口引脚信号如图 4 -14 所示（面对插口）：

GND	$\overline{WR}$	A3	A6	A2	A5	D0	D2	D6	D3	$\overline{IORQ}$	+5V	NC	NC	NC
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
GND	A7	A0	A4	A1	$\overline{RD}$	D1	D7	D5	D4	NC	+5V	NC	NC	NC

图 4 -14

内存扩充插口引脚信号如图 4 -15所示（面对插口）：

NC	+5V	$\overline{IORQ}$	$\overline{RD}$	$\overline{NMI}$	$\overline{WAIT}$	$\overline{M1}$	$\overline{RFSH}$	D7	D2	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	$\overline{RESET}$	GND
22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23
NC	+9V	NC	$\overline{WR}$	$\overline{MREQ}$	$\overline{HALT}$	$\overline{INT}$	D1	D0	A0	NC	D6	D5	D3	D4	ϕ	A15	A14	A13	A12	A11	GND

图 4 -15

(四) 扩充外设及接口

LASER 310系统对已开发的配套外设,均分配给16个I/O口地址:

00H～0FH　　打印机

10H～1FH　　软磁盘驱动器

20H～2FH　　游戏棒

30H～3FH　　调制解调器

70H～7FH　　内存扩充卡(分区切换用)

事实上一个外设并不需要这么多个口地址,所以可用不完全译码方式,以简化译码器结构。例如设备只需用一个地址时,以A7～A4四位地址信号即可选通这个I/O口。

用户自行扩充的I/O设备,最好使用上述以外的地址,以保持与系统的兼容性。

各种外设可能是截然不同的,限于篇幅,以较常用的打印机接口为例,说明扩充的方法。

计算机的打印输出设备主要有两类。一类是点阵打印机,其中最常用的是针式打印机。它按主机输出的字符代码控制打印头撞针的运动,通过击打色带,在纸上留下点阵构成的字符图形。另一种是X-Y绘图仪,它按输入代码控制绘图笔和卷纸轴的运动,以二者的合运动在纸上绘出向量图形或字符。LASER 310的标准设备是PP-40四笔绘图仪,一般也称作"四色打印机"。

打印机和绘图仪都属于智能终端设备。它们有自己的处理器,还有存贮驱动程序和字符点阵、向量数据的ROM。所以它们的机械动作并不需要直接由主机操纵。主机只需送出规定的字符代码或控制代码,机器就能自动完成所需的打印动作。

PP40用一片6805微处理器控制机器工作。接口的任务主要是将打印代码送给6805的数据输入端。为了使主机和打印机的工作保持同步,采取"查询"方式联络。打印机向主机反馈一个"BUSY(忙)"信号。主机每次送入打印数据前应查询打印机的状态。BUSY = 1表示以前输入的数据尚未处理完毕,主机应暂缓输出,在等待中反复查询;BUSY = 0表示可接受新的打印数据。

LASER 310的打印机接口PI-20的逻辑图见图4 -16。它由主板插头、译码器、锁存器、查询电路和打印机插头五部分组成。

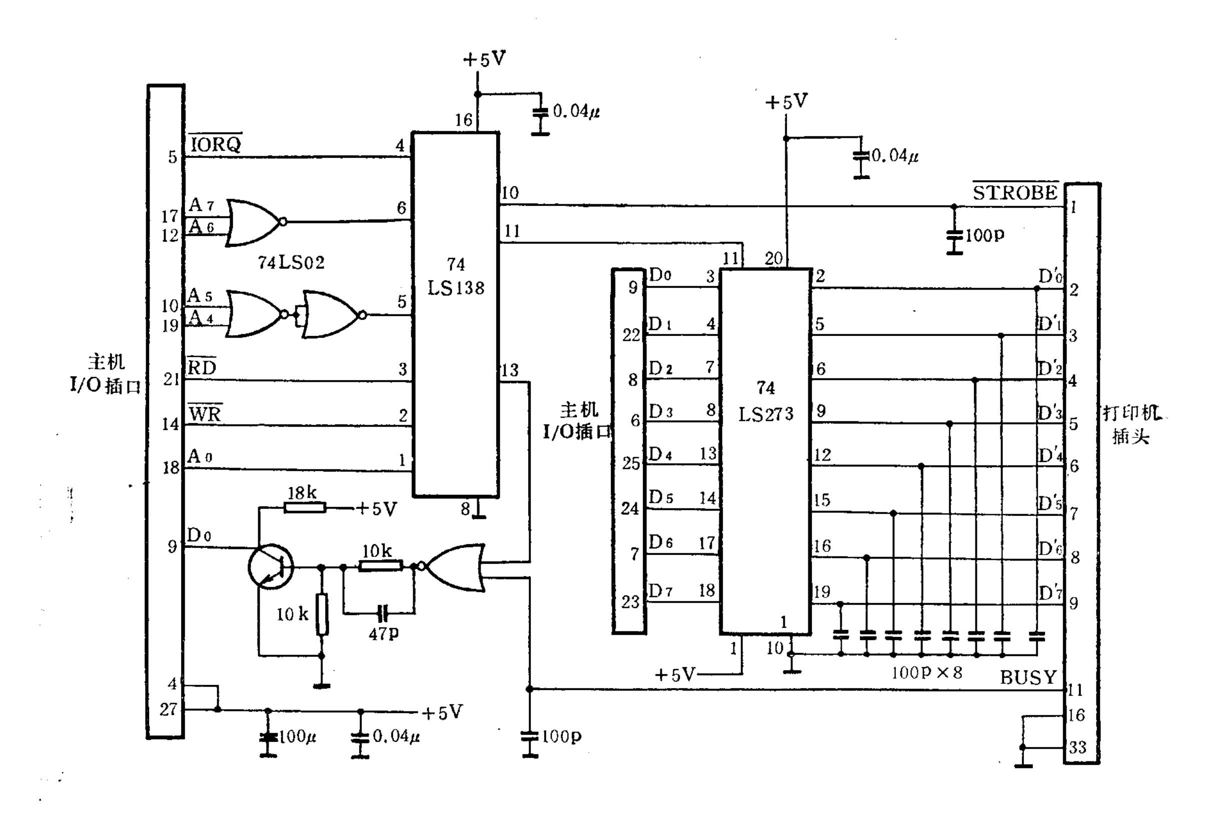


图 4-16　PI-20接口逻辑图

1．主板插口。在主机印板的 I/O 扩充插口中，打印机接口只联接 A7～A4，A0，D7～D0 $\overline{\text{IORQ}}$，$\overline{\text{RD}}$，$\overline{\text{WR}}$ 等16条总线。

2．译码器。主机对打印机的访问，需用两个地址。一个用于输出，一个用于查询(输入)。系统给打印机分配的口地址是00H～0FH，即 A7～A4 均为 0，现增加 A0 参加寻址，A0＝0(偶地址)时为输出口地址，A0＝1(奇地址)时为输入口地址。译码器以 74LS138 为核心，使用方式比较特别。将高四位地址信号经三个与非门合并起来，作为译码器的允许信号，即G1 ＝ $\overline{\text{A7}+\text{A6}}$，G2A＝IORQ，G2B＝A5＋A4。这样，只有在$\overline{\text{IORQ}}$有效，A7～A4均为 0 时译码器才工作。译码输入是 C＝$\overline{\text{RD}}$，B＝$\overline{\text{WR}}$，A＝A0。由 74LS138 真值表可知，当 A0＝$\overline{\text{WR}}$＝0，$\overline{\text{RD}}$＝1 时，第11脚有效，可用以控制打印代码送入锁存器。A0＝$\overline{\text{RD}}$＝1，$\overline{\text{WR}}$＝0 时，第13脚有效，可用以控制接收打印机的BUSY信号。而当A0＝$\overline{\text{WR}}$＝1，$\overline{\text{RD}}$＝0 时，第 10 脚的输出信号用作发给打印机的$\overline{\text{STROBE}}$(选通)信号，使其向接口取入打印代码。

3．锁存器。用一片八 D 锁存器 74LS273。将 74LS138 第 11 脚的输出信号作为锁存脉冲，在$\overline{\text{WR}}$无效后的脉冲上升沿，将主机数据总线 D7～D0 的打印代码锁存在其输出端，等待打印机取用。

4．查询电路。由一个或非门和一级晶体管反相器组成，输出数据直接送入主机数据总线的D0。对D0进行位测试，便可获知BUSY信息。或非门以一个输入端接受打印机发送的BUSY

信号，另一输入端由译码器13脚的查询允许信号控制。若查询允许信号无效，D0输入总是1。若查询有效，则D0 = BUSY。

5．打印机插头。除接锁存器8位数据线外，只有$\overline{\text{STROBE}}$和BUSY两条控制信号线。

从接口的结构可见，打印输出应分为三个步骤：一是查询打印机是否忙，忙则等待；二是输入打印代码；三是发出打印机选通信号。以上过程也就是用高4位为0的奇偶地址读写I/O口。以Z80汇编语言表达，则有

```
START:    IN  A,(01H)        ;查询打印机状态
          BIT 0,A            ;状态测试
          JR NZ,START        ;若忙则反复查询
          OUT(00H),D         ;不忙则送入新的数据
          OUT(01H),D         ;发出STROBE信号
          ......
```

当然，这个程序段还不够完善。当打印机发生故障时，查询结果总为1，主机因此会陷入死循环。所以两次测试间应增加检测BREAK键是否按下的指令。

虽然绘图仪和行式打印机的工作原理并不相同，但PP-40的字符代码和主要控制代码同通用打印机是兼容的，PI-20接口的打印机插头也和EPSON打印机插口兼容。所以，LASER 310完全可以配接CP-80、FX-100等针式打印机，用于ASCII文本打印。如果没有PI-20接口，自行按图4-16组装一个也不困难。不过，点阵打印机和向量绘图仪的绘图控制是完全不同的，系统原来的打印驱动程序不适用于点阵式打印机的图形方式。若需用它来绘图（包括打印汉字），需要另行设计驱动程序。

四、视频电路

显示器件是微型计算机最基本的输出设备。最简单的是用发光二极管阵列或荧光数码管来显示数字。一般通用型微机的显示器件有液晶显示屏和阴极射线管（CRT）显示器两类。目前已普及化的液晶显示屏为单色、小面积，接口硬件简单，成本低，但显示方式单调，仅适合以数据运算为主的袖珍机型。多数微型机均采用通用型的CRT显示器和普通电视机作为显示终端设备，利用其大屏幕、多彩色的特性，支持文本编辑、人机对话、图形游戏等功能，充分发挥计算机的效益。CRT显示器和电视机，特别是彩显、彩电，需要输入复杂的视频信号才能按要求进行工作，因而，产生视频信号的电路成了计算机中比较复杂的一个模块。从系统结构上看，视频电路是CPU同显示器之间的接口，对它的剖析需要计算机和电视两方面的知识。考虑到计算机爱好者不一定都懂得电视原理，本书将对其作一些简要说明。但限于篇幅，不可能向初学者详尽地讲解，需进一步了解的读者请阅读电视专业书籍。

（一） 视频显示原理

1．CRT显示和电视信号。

计算机显示器与普通电视机都使用阴极射线管作为显示器件。它有一块涂布荧光粉的屏幕，受电子轰击便会发光。电子枪发射的电子束聚焦到荧光屏上便产生一个光点，它的亮度由电子束的强弱决定。电子枪外面围有行、场偏转线圈，在锯齿波电流的作用下分别产生水平、

垂直方向偏转运动的磁场，控制电子束在屏幕上一行行地由上而下地扫描，形成由光点组成的光栅。如同用黑白点在纸上印出画面一样，光点的明暗分布在屏幕上构成图像。每行光点扫描完毕，电子束截止并偏转到下行始端；每幅光栅扫描完毕，电子束截止并偏转到屏幕上端，周而复始地循环进行。若每秒扫描形成25帧图像，根据人眼视觉暂留原理，则可用来显示活动画面。

彩色显像管的主要不同之处是荧光屏上交错分布着发红、绿、蓝光的三基色荧光粉点，电子枪发出的也是三条电子束，分别轰击对应基色的荧光粉。由于人眼的分辨力关系，三个光点被视为一体，三基色的不同强度便合成了各种特定的色彩。

除一些高级的计算机专用显示器外，以通用视频显示器或电视机作为显示设备时，其视频信号必需同普通电视标准兼容。目前世界各国采用的电视信号标准有多种。黑白电视分 A、B、D、E、G、I、K、L 等制式，彩色电视不但分别同上述某种黑白制式兼容，其彩色制式又分为NTSC（平衡正交调幅制）、PAL（平衡正交调幅隔行倒相制）和 SECAM（调频行轮换制）三种。每种制式的技术标准各异，互不兼容。所以显示器和电视机只能正确处理制式相符的电视信号。我国的黑白电视标准采用DK制，彩色电视制式为 PAL 制。所以使用我国标准的显示设备，应给它提供符合 PAL-DK 制的电视信号。LASER 310 有分别适用于三种彩色制式的机型，国内引进或组装的都是适用于PAL-DK制的。

计算机的视频电路，原理上类似于数字式电视摄像机的后半部分（有图像输入和处理功能的计算机，则相当于完整的摄像机）。这台“摄像机”的前端部分由人取代。当需要在显示器荧光屏上显示一幅图画时，人的眼睛便代替摄像机镜头，将图画摄入大脑，在这儿将图像的光信号，即亮度和色彩的分布数字化（当然，并不一定面对一幅真实的图画，也可以通过形象思维甚至抽象思维来达到这一步），再经由手-键盘-数据总线-CPU-接口电路的途径，实现“光电转换”并存入显示存贮器（VRAM）。VRAM 相当于摄像管的“靶”，上面的信息分布是与“镜头”前的景观对应的。视频电路如同摄像机一样地对“靶”进行扫描，取出图像信息，合成电视信号输出。当然，计算机与摄像机除了有“大同”的一面，由于自身的特点也有“小异”的一面。下面就彩色全电视信号的组成着重从二者异同的角度进行讨论。

彩色全电视信号由色度信号 F，亮度信号 B（或用 Y 表示），复合消隐（包括行消隐和场消隐）信号A，复合同步（包括行同步和场同步）信号 S 等迭加在一起组成，一般用 FBAS 表示。PAL 制彩色全电视信号编码过程见图 4 -17。此外，电视广播信号中，还同时传送一路调频的伴音信号。有的计算机射频输出亦有一路伴音，但一般都只将声频信号直接驱动机内扬声器发声。

2. 同步信号。

为了使摄像端和显像端的扫描过程完全同步，以保证画面位置的准确，电视信号中必须包含同步信号。同步信号分为行同步和场同步两种，都是负脉冲方波，与图像信号不会混淆。同步脉冲分别经过显示器的微分、积分电路，产生行、场锯齿波，加在偏转线圈上以控制电子束扫描行程，使每个明暗光点和每条扫描线都落在特定位置。

DK 制规定，每秒扫描产生 25 帧画面，每帧画面由 625 行水平扫描线组成，行频为625×25＝15625Hz。标准电视系统为了减少画面闪烁和减小视频带宽，采取隔行扫描的方式，每帧图像由隔行扫描的两场合成，奇数场扫描第 1 、3 、5 ……行，偶数场扫描第 2 、4 、6 ……行，

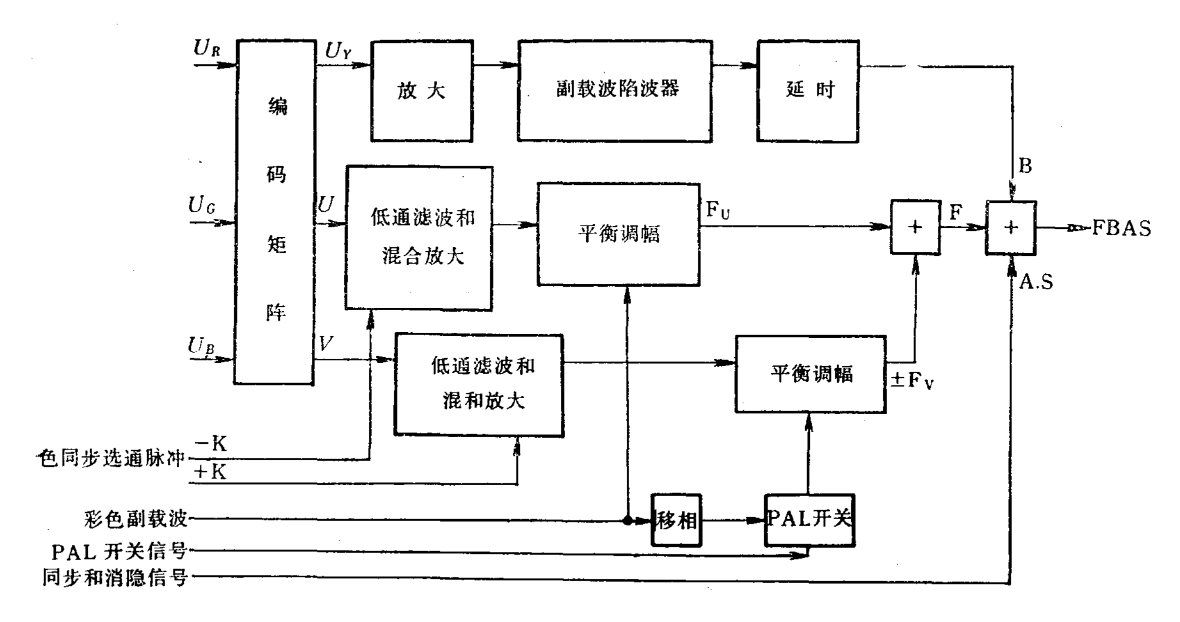


图 4-17 彩色全电视信号编码流程图

每场扫描 312.5 行，场频 50Hz。

计算机显示画面也有特定的屏幕位置坐标，也需要含有同步信号。若采用隔行扫描方式，一帧图像中的每个字符都要由两场中的不同部分合成，并且视频时序也要复杂一些。所以普及机的视频系统常采用逐行扫描方式。为了同普通电视兼容，场频行频不变，每场取整数行，虽每场仍按隔行扫描，但两场扫描线完全重合，一场就是一帧。这样，垂直方向的分辨率也就降低了一半。不过对于普及机，已基本够用。

3. 消隐信号。

消隐信号用来在行、场扫描完毕、偏转方位转回始端的过程（回扫）中使电子束截止，以免屏幕上出现回扫线。消隐电平较图像信号电平为负，位置在同步脉冲的两侧。电视机设计时，考虑到设备老化或电压不稳等会造成画面幅度缩小，因而采取了“过扫描”措施，使扫描线超出实际屏幕的边框。这对于主要信息分布在屏幕中间的模拟性的电视画面是没有什么影响的。但对计算机的字符数字画面，过扫描却可能丢失信息。因此，计算机视频显示系统常将行、场消隐脉冲设计得较宽。这样，就使有效画面缩小，可能在屏幕上形成边框。

4. 图像信号。

电视图像信号是客观实体的光信息在摄像机内转化成的电信号，是连续的、变化不规则的模拟信号。而计算机的图像信号是以数字方式设定的。对于明暗两种光点组成的字符信息，只需固定的高/低电平两种信号便足以表达。计算机图形信息，是将数字信号转变为模拟信号输出，以产生不同层次和色调变化。数-模转换中，数字信号的位数越多、频率越高，所得到的模拟信号质量也就越好。但由于计算机字长、时钟频率、显示存贮器容量等限制，即使一般高档机也难于产生出能同电视画面媲美的细腻图像来。尤其是低档的学习机，仅能用很少的几位数据定义图像信号，因而转换成模拟信号波形也是台阶式的，只能显示出有限的几种灰度和色调。所以，只能算是“数字式模拟信号”。

黑白电视图像信号就是亮度信号。显示器件根据亮度信号，控制电子束的强弱，从而使扫描线上各光点的亮度与发送端一致，正确地再现图像。彩色电视图像信号原则上应是代表三基色分量的三个亮度信号，用以分别控制三条电子束的强度。同时，电视广播又要求彩色电视信号必须让黑白电视机也可收看。所以，其中还应包含一个相当于黑白电视图像信号的亮度信号。这样，就需要同时传送四路亮度信号了。

对于通讯技术来说，电磁波信道是一种有限的资源，为满足多方面需要，只能有计划地节制使用。黑白电视的一路亮度信号需占用频带宽度为 4～6MHz（我国为 6MHz）。若上述四路信号同时传送，信道是容纳不下的。现代彩色电视技术所解决的核心问题之一，就是设法把四路信息巧妙地“揉”在一起，仍然用一路黑白电视的带宽传送，而又能让相互的干扰减小到可容许的程度。具体说，就是用一个亮度信号和两个色差信号来表示上述四个信号，两个色差信号经正交调制后叠加成一个色度信号。色度信号再插入亮度信号的频谱间隙中，于是四路变一路，带宽仍然只占 6MHz。

（1）亮度与色差信号的产生。任何色彩的光均可分解为红(R)、绿(G)、蓝(B)三基色，但它的总亮度并非三基色亮度的平均和。它们的关系可用“亮度方程”表示。

$Y=0.30R+0.59G+0.11B$。

由上式可以看出，用其中三个量便可决定另一个量。所以彩色电视信号可由一个亮度信号和两个色度信号来传送。出于多方面的技术考虑，实际应用时选择了亮度信号 Y 和色差信号 B-Y、R-Y。“色差信号”就是基色信号电平与亮度信号电平之差，因而可能有负值。由亮度方程可以导出：

$B-Y=-0.30R-0.59G+0.89B$，

$R-Y=0.70R-0.59G-0.11B$。

以上三式右边各项系数都小于 1，所以在摄像机中，将三支摄像管输出的基色电平通过简单的电阻矩阵，各按比例衰减，便可分别按上式合成 Y、B-Y 和 R-Y 三个电平信号。

MC6847 是根据显示数据所含的色彩代码，用译码器产生出相应的 Y，B-Y 和 R-Y电平数据，再经 D-A 电路转换为模拟电平，供彩色编码使用。

（2）正交平衡调幅和彩色编码。彩色信号要调制在高频电磁波上传送。这个电磁波称为副载波。“副”是对于电视广播的射频主载波而言。NTSC 和 PAL 制都采用正交平衡调制法。正交就是两个色差信号分别调制在频率相同、相位相差 90° 的副载波上，再叠加在一起。平衡调幅就是调制后抑制掉调幅波的载波分量，只保留携带信息的上下边带。两个平衡调幅波叠加的结果，便产生一个新的调幅波，一般称为色度信号。它的幅度和相位包含着两个色差信号的幅度和极性信息，用同步检波的方法便可以分别检出。以上过程为“彩色编码”。

（3）频谱交织。由于亮度信号频谱中有很多间隙，若副载频选择得正好处于这种间隙的中间，那么已调制的色度信号就可以根据“频谱交织”原理，在叠加时插入亮度信号频带中而不致互相干扰。副载频一般选择 1/2 行频的奇数倍。NTSC 为 3.58MHz，PAL 为 4.43MHz。

（4）色同步信号。为了压缩带宽，在平衡调幅时抑制了副载波，但接收端对平衡调幅波进行同步检波时，又必须使用调制时的副载波。为了传送副载波信息，在行同步脉冲后肩的行消隐期内，即一行图像信号之前，发送约 10 个周期的副载波“样本”，供接收端了解其频率和相位，以便“仿制”出与发送端同频同相的副载波来。这个副载波群称为“色同步信号”。在相应

时刻往色差信号中加入一个 10 倍副载波周期宽度的脉冲(色同步选通脉冲)，调制器对副载波平衡调幅时，便会在这一位置上留下所需的色同步信号。

PAL 制式的特点是色度信号逐行倒相，使显示系统的相位失真在两行之间互相对消，以正确重现彩色。这种逐行倒相的信息，也通过色同步信号在倒相行相位变反来表示，供接收端识别处理。

经彩色编码形成的色度信号，再同复合同步信号、亮度信号混合放大，成为彩色全电视信号，可直接送到显示器或电视机的视频输入口。若需由电视机天线插口接收，还要经射频(RF)调制器将视频信号调制到甚高频段或超高频段的主载频上。

在接收端，若是黑白机，只有亮度信号起作用，频谱间隙的色度信号只形成轻微的干扰网纹。彩色机中则将图像信号分作两路，一路经过副载波陷波器后主要留下亮度信号，一路用副载频进行同步解调分离出两个色差信号。然后，通过解码矩阵还原出与摄像端关系相同的三基色电信号，分别控制三个电子束轰击荧光屏上的三色荧光粉，按摄像端的色彩和亮度显示出来。

(二) **VRAM 6116**

6116 属于静态 RAM，容量 16Kbit。存贮单元电路由六个 MOS 管组成。内部结构见图 4-18。

在系统中它被用作 CPU 与视频显示发生器间的缓冲存贮器。CPU 向它存入需显示的字符和图形代码，视频显示发生器再取去合成视频信号。因此，它必须接受双方的访问。一般来说，应该用多路转换器对双方的地址和数据信号进行切换。

LASER 310 采取了一种简化的结构。它的11条地址线(DA10～DA0)既通过电阻同地址总线相连，又直接连接视频显示发生器 MC6847 的地址线。这样，6847 的地址信号便处于

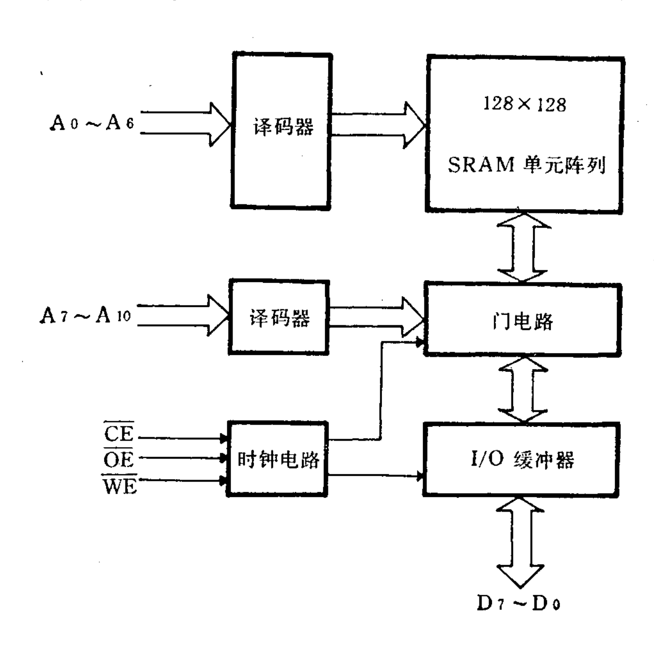


图 4-18 6116 逻辑图

优先地位。当6847发出一组地址信号时，CPU地址总线的状态对6116是无效的。因为隔离电阻的关系，若某一地址线上，6847的信号和地址总线的信号相反，则6116端口得到的总是6847的信号电平。当CPU需访问VRAM时，GA003的$\overline{VR}$信号便迫使6847的地址输出端口进入高阻态，于是6116地址端得到的便只是CPU的地址信号了。

6116的八条数据线(D7′～D0′)，一方面直接连接6847的显示数据输入端DD7～DD0，一方面连接GA004中的八总线收发器。在CPU不访问VRAM时，因总线收发器呈高阻态，6116只能向6847提供显示代码。而当CPU对它进行读写时，数据却可能串入6847数据输入口。这是系统设计必须考虑的问题。

因为视频电路对6116的访问是周期性和持续的，所以它的片选信号端$\overline{CS}$和读允许信号端$\overline{OE}$都接地，使在任何间刻都能读取数据。写允许信号$\overline{WE}$由GA004发出的写VRAM信号$\overline{VRWR}$控制。此信号是低电平时为写有效，高电平时为读有效。

(三)　**视频显示发生器(VDG) MC6847**

视频显示发生器是视频电路的核心器件。它的任务是按系统和用户对显示模式的设定，按一定时序扫描VRAM，从中取出显示码，据以生成电视信号的各种必要成分。

MC6847是美国Motorola公司生产的大规模集成化VDG芯片，功能较完备，可供各种计算机系统使用。MC6847为逐行扫描型，MC6847Y为隔行扫描型。芯片电路为兼用结构，改变内部几个控制电平即可变型。它们均适用于美国的NTSC-M制电视标准。LASER 310采用MC6847P。后缀“P”表示芯片为塑料封装。

6847有三个数字式综合模拟量输出端：Y为亮度、消隐和复合同步信号；ϕA为色差信号R-Y，ϕB为色差信号B-Y和色同步选通脉冲。

因为它的输出尚未经彩色调制，所以配合适当的外围电路也可用于PAL制式。同时，6847在黑白制式方面是采用美国的M制，场频60Hz，行频525Hz。变为我国的DK制电视信号，也需要一些外围电路来进行调整。LASER 310的视频电路是将MC6847用于PAL-DK制的一个很好的例子。

丰富的显示功能是LASER 310取得成功的重要因素，而这又主要归功于6847。目前国内书刊上尚未见到对6847的详细介绍。所以我们讨论得充分一些。MC6847引脚见图4-1，图4-19是MC6847内部逻辑框图(附书末)。

1．时钟

6847要求一个外部单相时钟信号CLK，应为频率等于NTSC制副载频3.58MHz的方波，占空比50%左右。芯片的各种脉冲信号都在它的基础上产生。LASER 310-PAL型机使用的是其近似值——17.7MHz系统时钟信号的5分频，3.547MHz(它同时也是Z80A的时钟信号ϕ)。本节涉及视频的具体时间时，我们均按6847标称值，请读者注意。

LASER 310为改变制式，调整场频，对CLK的输入进行控制，必要时可令时钟“停摆”。

单相的主时钟在VDG内转变为二相时钟ϕ1、ϕ2，二者频率相同，相位相反，用作控制图像数据输出的时钟脉冲，交替触发，使每个CLK周期内可产生两个光点信号，相当于7MHz时钟频率的效果。同时，CLK还经3.5分频成为1.02(LASER 310-PAL系统实为1.01)MHz的时钟信号，供同步计数器使用。这一点上，它与中华学习机不谋而合。

2．显示模式。

显示模式的丰富多彩是 6847 的一大特色。它的显示模式分字符数字、半图形和图形三类。字符和半图形模式中，使用内部字模和外部字模的显示模式又有不同，字符还可作正/反相显示选择。图形模式分为 8 种分辨率。屏幕显示的背景色可有两种选择，字符或图形的彩色种数随模式而异，可显示色彩有 8 种(不包括黑色和某些深浅效果)。6847 的显示模式用 8 个输入控制信号设定。其中 7 个见表4-1。

表 4-1

$\overline{A}$/G	$\overline{A}$/S	$\overline{INT}$/EXT	INV	GM2	GM1	GM0	模式内容	色彩
0	0	0	0	×	×	×	内部字模字符正相显示	2
0	0	0	1	×	×	×	内部字模字符反相显示	2
0	0	1	0	×	×	×	外部字模字符正相显示	2
0	0	1	1	×	×	×	外部字模字符反相显示	2
0	1	0	×	×	×	×	4单位半图形 (SG4)	8
0	1	1	×	×	×	×	6单位半图形 (SG6)	8
1	×	×	×	0	0	0	64×64彩色图形模式 1 (CG1)	4
1	×	×	×	0	0	1	128×64分辨率图形模式 1 (RG 1)	2
1	×	×	×	0	1	0	128×64彩色图形模式 2 (CG 2)	4
1	×	×	×	0	1	1	128×96分辨率图形模式 2 (RG 2)	2
1	×	×	×	1	0	0	128×96彩色图形模式 3 (CG 3)	4
1	×	×	×	1	0	1	128×192分辨率图形模式 3 (RG 3)	2
1	×	×	×	1	1	0	128×192彩色图形模式 6 (CG 6)	4
1	×	×	×	1	1	1	256×192分辨率图形模式 6 (RG 6)	2

注：$\overline{A}$/S 和 INV 可对每个显示位置(32×16)分别控制，方法是将控制信号含于显示数据中，并将相应的数据线连接该控制端，LASER 310 将显示数据线DD7 接$\overline{A}$/S，DD6接INV，因而可指定每个显示符号的类型。

CSS 显示背景色选择。各种模式都有两种背景色可以选择。

LASER 310中 $\overline{A}$/G 由 GA004 的Q3控制，CSS 由 GA004 的 Q4 控制。GM0 和 GM2 接地，GM1接 +5V高电平，因而图形模式只有 CG2一种〔即 MODE(0)〕，别无选择。

3. 对显示 RAM的寻址访问。

6847 可用 13 条显示地址线访问 VRAM，地址线为三态输出。

各种显示模式的分辨率和彩色不同，显示的信息量多少不一，因而一屏图像所需的 VRAM 空间也就有差别。见表 4-2。

LASER 310 仅配置了 2KB 的 VRAM，所以 DA11和 DA12悬空未用。

视频显示一般为内存映像方式，屏幕上的每个像素都是同显示 RAM 里的每个单元或单元里的每位数据一一对应的。这就要求对屏幕的电子扫描与对 VRAM 的地址扫描严格同步进行。对 VRAM 的访问以字节为单位，对屏幕的扫描以光点为单位，在不同的显示模式下，二者有着不同的对应关系。这是 VDG 面临的主要逻辑任务之一，由显示模式译码器的输出进行控制。因为 VDG 内部各种时序都共同建立在主时钟 CLK 的基础上，因而可以在复杂的变化中协调一致。

表 4-2

模式	分辨率	像素数据	需要VRAM容量	地址范围	地址线
A&S	32×16	8 位	32×16=512B	0000H～01FFH	DA0～DA8
CG1	64×64	2 位	16×64=1024B	0000H～03FFH	DA0～DA9
RG1	128×64	1 位	16×64=1024B	0000H～03FFH	DA0～DA9
CG2	128×64	2 位	32×64=2048B	0000H～07FFH	DA0～DA10
RG2	128×96	1 位	16×96=1536B	0000H～05FFH	DA0～DA10
CG3	128×96	2 位	32×96=3072B	0000H～0BFFH	DA0～DA11
RG3	128×192	2 位	16×192=3072B	0000H～0BFFH	DA0～DA11
CG6	128×192	2 位	32×192=6144B	0000H～17FFH	DA0～DA12
RG6	256×192	1 位	32×192=6144B	0000H～17FFH	DA0～DA12

例如，在字符方式下，每个字符代码占 VRAM 一个字节，32 个字节对应屏上一行字符，而屏上每个字符由 5 × 7 光点矩阵组成(加上字符上下左右的空白，在屏幕上占 8×12 点位置)。所以从 VRAM 取得一个代码并向屏幕输出它所代表字符的一排 8 个点后，就要访问 VRAM 下一单元……直到一行字符的同一排光点(即屏幕上的一个水平扫描行)输出完毕，再重新访问 VRAM 中该行的第一单元，开始输出第二排光点的循环。反复扫描 VRAM 同一行 12 次，同时在屏幕上扫描输出 12 行，便显示出一行字符。然后再开始访问 VRAM 的下一行代码。

显示地址发生器由水平地址计数器和垂直地址计数器构成。地址就是它们在时序控制下的脉冲计数值，所以有逐一递增和循环往复的特点，与 CPU 的地址信号由指令给出的方式完全不同。地址计数器是串行输入、并/串行输出的二进制计数器。水平计数器共五位，Q0～Q3 输出DA0～DA3 地址信号，Q4 通过或门接DA4。垂直计数器共 9 位，Q1～Q8 输出 DA5～DA12地址信号，Q0也通过或门接DA4。地址计数脉冲由控制电路按视频时序发出。

(1)水平访存时序。计数脉冲频率按不同模式分为两种：

①长周期时序。每 8 个 CLK周期(以 1/f 表示)产生一个计数脉冲，使水平地址DA0～DA3增 1，各位输出的值在 0～15 之间变化。Q4 被置 0，DA4 由垂直计数器的 Q0 控制。当 DA4～DA12不变时，DA0～DA3 用来反复访问 VRAM 中连续的 16 个单元。长周期用于每显示行 16 字节的模式(CG1、RG1、RG2和RG3)。因为这类模式每一位显示数据对应于屏幕上的两个光点，所以把地址保持时间拉长以便与视频输出时序一致。

②短周期时序。每 4 个 CLK 周期产生一个计数脉冲。这时垂直计数器 Q0的输出端被置 0，DA4由水平计数器的 Q4 控制。DA0～DA4地址值为 0～32，用于每显示行 32 字节，每位数据对应于一个光点的模式(字符、半图形、CG2、CG3、CG6 和 RG6)。

(2)垂直访存时序。按不同模式显示像素的垂直结构，分为四种。字符、半图形的垂直计数脉冲是行预置脉冲 $\overline{RP}$。$\overline{RP}$ 由行频经 12 分频产生。图形模式的垂直计数脉冲按每像素的扫描行数(192/垂直分辨率)分别为行频、1/2 和 1/3 行频脉冲。

当水平时序为长周期时，垂直计数器 Q0 输出经或门送入 DA4。水平短周期时 Q0 置 0，

垂直计数脉冲直接送入垂直计数器的第二个数据输入端，同时也就由 Q1 送入 DA5。

模式控制将分别对DA9～DA12输出置 0，以限制访存范围。场同步脉冲使显示地址计数器全部清零，以便重新开始计数。

存贮器选择信号$\overline{MS}$用作显示地址线门控信号，当 CPU 需要访问 VRAM 时，只要使此信号变低，便可让各显示地址输出端进入高阻态，VGD 同地址线隔离。LASER 310 用译码器输出的$\overline{VR}$信号控制$\overline{MS}$，使 CPU 访问期间独占 VRAM 地址线。不过，$\overline{MS}$并不改变 VGD 的时序，它仍照常工作。当 CPU 写 VRAM 时，VGD 从数据线取入的不是 VRAM 中的数据，所以屏幕相应位置显示混乱，形成闪烁光斑。因而 CPU 对 VRAM 的访问应尽可能利用 VGD 非访存期间进行。

4. 同步时序。

同步信号发生器是 VDG 中一个独立模块，它对时钟脉冲计数，生成同步和控制信号，不受其他输入信号的影响。同步信号发生器由两个线性移位寄存计数器分别对各自的时钟脉冲计数，计数值送入时序译码器。译码器按既定的"时刻表"和两个计数器的"钟点"数，发出时序控制信号，协调 VDG 各部分的工作，参加图像信号的合成，有的并经引脚输出。

(1)水平时序。水平计数器由七位构成，只用来从 0 计数到 64，然后复零重新计数。计数脉冲是 1.02MHz 时钟信号，65 个周期宽度为 $65\times1/1.02\approx63.5\mu s$。这便是扫描一行的时间，行频为 $1/63.5\approx15746$Hz，符合 NTSC-M 制要求。一个同步计数信号宽度为 $3.5\times1/f$，每行也就是 227.5 个 $1/f$。时序译码器按计数值发出水平扫描行程中各个阶段的定时信号，以控制输出所需的时序信息。一个水平扫描行可分为七个部分，见图 4-20。各部分的定时(折合为以 $1/f$ 为单位的数值)如下：

a. 行同步脉冲 17.5。

b. 同步脉冲后肩(消隐期)17.5(其中行同步脉冲至色同步选通脉冲3.5)。

c. 色同步选通脉冲10.5。

d. 左边框29.5。

e. 图像行128。

f. 右边框28。

g. 同步脉冲前肩(消隐期) 7 。

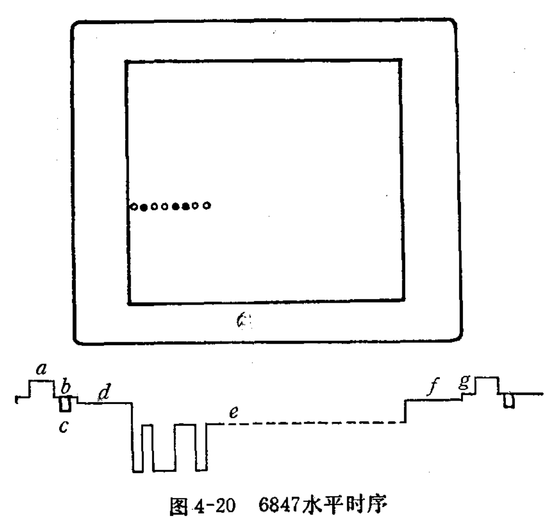


图 4-20 6847水平时序

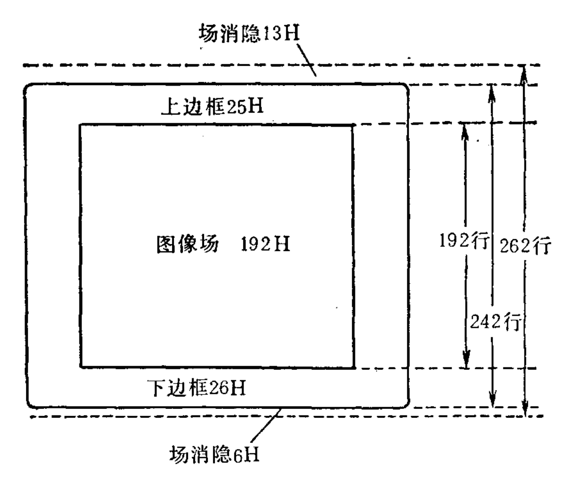
图中画在屏幕范围外的部分实际是在电子束回扫逆程中，两端连起来看就是一个消隐脉冲上面迭加了行同步脉冲和色同步选通脉冲。图像行为 $128\times1/f$。因为图像信号由两相时钟 $\phi1$ 和 $\phi2$ 交替移位输出，每个 $1/f$ 期间可输出两个光点信号，所以一行可显示256个光点。这便是 6847 的水平分辨率。各种模式不同的图形分辨率都建立在这同一的硬件分辨率的基础上。

水平时序的译码输出信号作用是：第一，视频输出电路控制与合成视频信号；第二，行同步

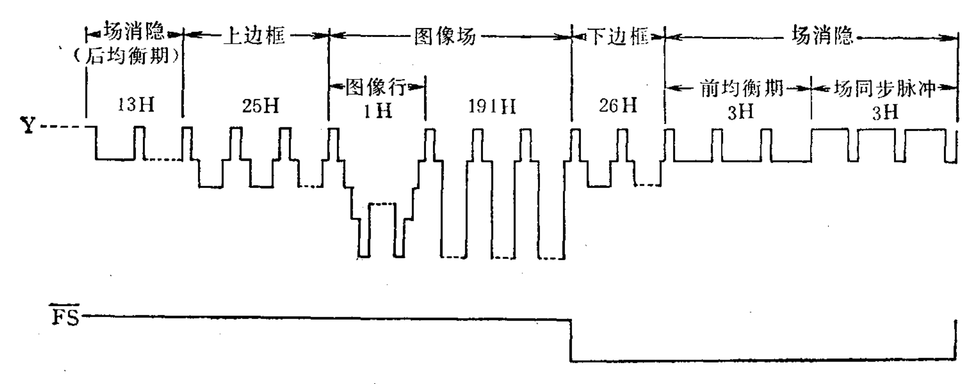
信号通过行计数器产生垂直访存地址计数器的计数脉冲，经过计数器还产生内部字符ROM的行地址；第三，行计数器计数12个行脉冲后复位，并输出行预置信号$\overline{RP}$；第四，向外部输出行同步信号$\overline{HS}$，与$\overline{RP}$一起供外部字符发生器的行计数器使用。

（2）垂直时序。垂直时序计数器为10位，用以从0计数到523，524＝0。计数脉冲为行频的二倍31.468kHz。每个周期31.7783μs。选取它而不是直接使用行频脉冲，是因为若电路接成6847Y时，隔行扫描方式的每场有一个半行，需要宽度为1/2行周期的脉冲进行均衡。对于6847不存在这个问题。6847的场周期为31.7783μs×524＝16.65ms，场频1000/16.65＝60Hz。

6847的垂直时序见图4-21，*a*是屏幕显示的垂直结构，*b*是脉冲波形。*a*中屏幕范围外的部分实际是在电子束由屏幕右下角返回左上角的回扫逆程中。为了衔接隔行方式下每场的一个半行，场同步脉冲两侧的消隐期内均为H/2宽度的"均衡脉冲"。场同步脉冲宽度为3H，为了在它的周期内不致使行振荡器失去同步信号，在它上面开了很窄的"口子"，以便仍然能够分离出三个行频脉冲来。



a. 6847垂直时序示意图



b. 6847垂直时序波形

图4-21

垂直时序信号经译码后用于:第一,视频输出电路控制与信号合成;第二,场同步信号作为显示地址计数器和行计数器的清零脉冲;第三,向外部输出垂直同步信号 $\overline{FS}$。

$\overline{FS}$ 与视频信号中的场同步脉冲完全不同。它是在一场的图像行传送完毕,开始进入下边框时变低有效的,在场同步脉冲后沿重新升高。它开始有效的时刻正值 VDG 停止访问 VRAM,所以用来表示 CPU 可占用 VRAM 地址线的时刻。从 $\overline{FS}$ 有效到上边框结束,共经过 3+3+25+26+3=60H (行周期),历时约 3.8ms。这便是 CPU 可利用的时间。LASER 310 增加调整时序以后,进一步延长了这一时间间隔。

5. 显示信息。

6847 的显示数据由 DD7～DD0 八条数据线输入,其中包含 光点信息和色彩信息。在不同模式下,显示数据的意义和作用是不同的。

(1)光点数据。

①内部字符模式:DD5～DD0作为内部字符 ROM 的列地址,行地址是行计数器的计数值,用来寻址 64 个字符点阵数据存贮单元。ROM 的矩阵结构是 64 行×5 行,每列 8 位存贮单元电路,共存有 64 个 5×8 点阵的 ASCII 字模,如图 4-22 所示。

D5-D4＼D3-D0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	↑	←
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	;	:	<	=	>	?

图 1-22

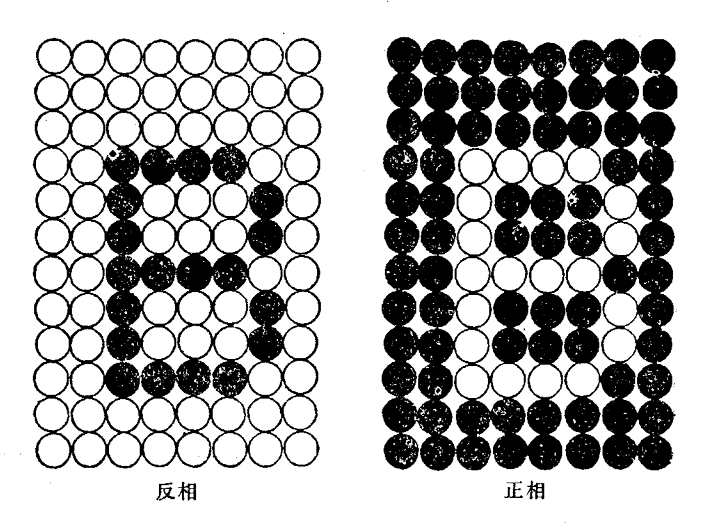


图 4-23 内部字符点阵结构

字符发生器的输出端加有逻辑控制，自动在点阵数据输出前后送出三个和二个空行作为行间隔，当INV(DD6)为0时，使数据反相后输出，因而共可显示128种字符。字符的光点矩阵模型见图4-23。可以看出，LASER 310的基本显示方式——绿底黑字属于“反相”字符，而通常称为“反白”的才是正相。这种设置是系统软件规定的。

②外部字符模式：显示数据线DD7～DD0用来接受外部字符发生器送入的一行光点数据。

外部字符发生器ROM应为8×12点阵的64单元结构。它接受显示数据线的数据作为列地址，用一个四位计数器对水平同步信号$\overline{HS}$计数以产生点阵的行地址，此计数器由行预置信号$\overline{RP}$清零。读出的一行八位点阵数据送入VDG的DD7～DD0。显然，系统的数据线先用来将VRAM的数据送给外部ROM，再用来将外部ROM的数据送入VDG，所以需用门电路进行切换(图4-24)。外部字符可由用户自行设置为英文小写字母、其他文字或特殊符号。LASER 310中此脚接地，故只能显示内部字符。

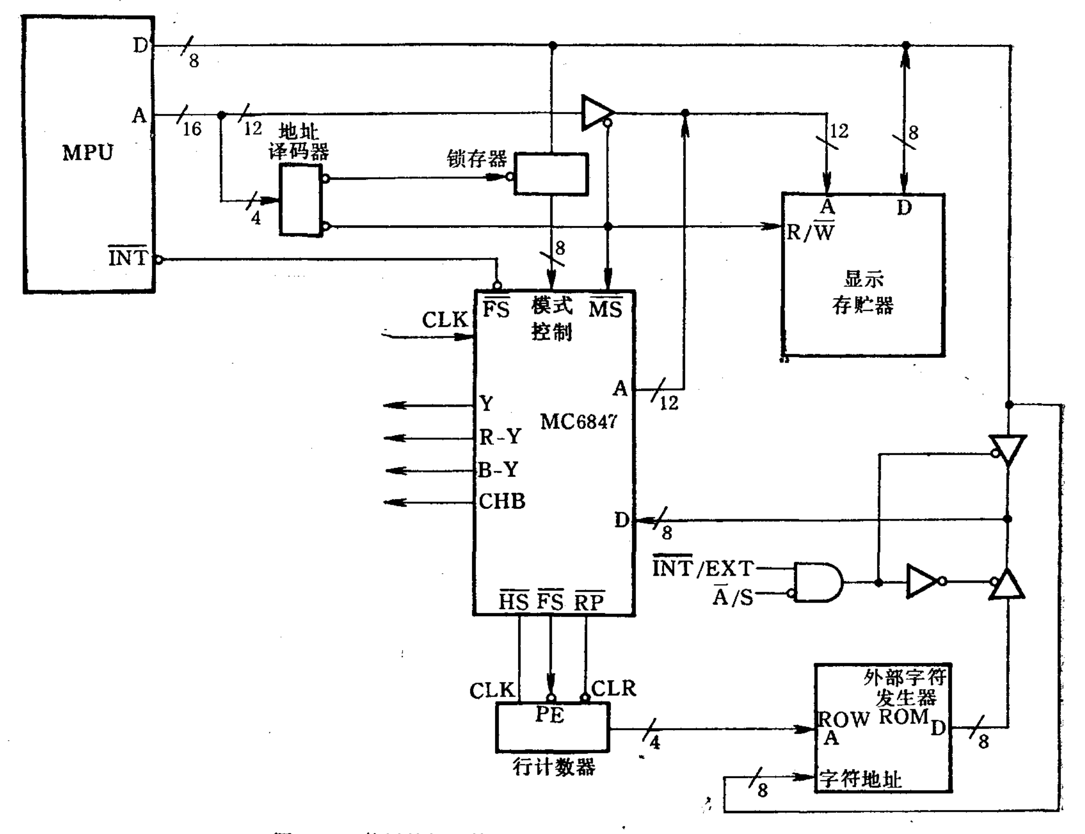


图4-24 使用外部字符发生器的电路

③SG4模式：当$\overline{INT}$/EXT＝0时显示数据的DD3～DD0作为SG4图符发生器ROM的列地址信号，寻址16个图符点阵数据存贮单元。这些图符如图4-25所示。DD3～DD0与图符结构对应关系如图4-26所示。

④SG6模式：当$\overline{INT}$/EXT＝1时，并非从外部取得图符点阵数据。DD5～DD0作为内部SG6图符发生器的列地址，寻址64个图符单元。数据各位与图符结构对应关系如图4-26所示。

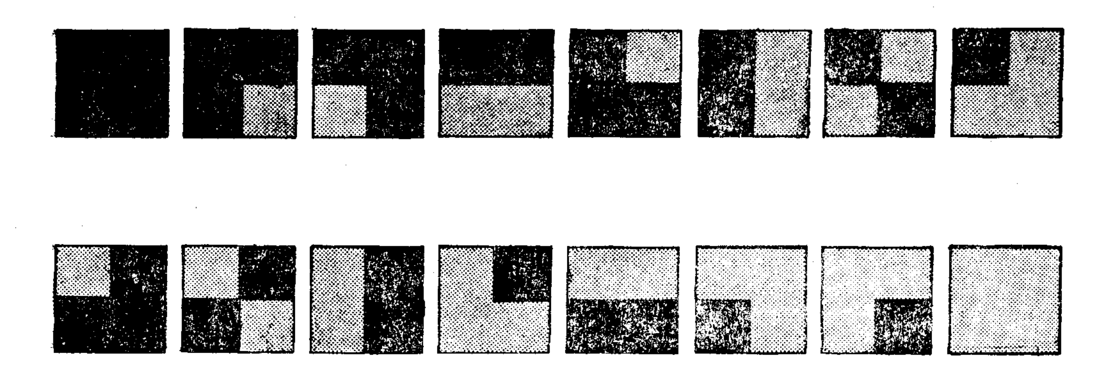


图 4-25　内部图符

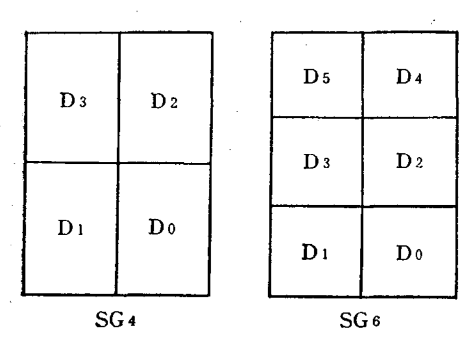


图 4-26　内部图符数据结构

⑤图形方式：对于一位像素的模式，每位显示数据都直接作为光点信息。水平分辨率是256点时，每位表示一个光点。而水平分辨率128点时，每位表示相同的两个光点。对于两位像素的模式，显示数据全部表示色调信息，光点信号由控制电路全部给出"1"。

来自不同数据源的光点数据，由模式控制多路选择器选出所需的一组，并将奇偶位分为两组四位数据分别并行送往两个视频移位寄存器。两相时钟$\phi1$、$\phi2$分别控制从两寄存器输出端攫取数据，形成二倍CLK钟频的串行数据。同时两相时钟又经控制逻辑电路，形成含有送入控制和时钟禁止信息的$V\phi1$、$V\phi2$，控制移位寄存器送入和移位的速度。若送入频率是读出频率的一半，则每一位数据将被重复读取一次，以适应有的模式中一位数据产生两个光点的需要。

（2）色调数据。显示色彩数据是同光点数据分别传送的。在输入的显示数据中含有色调代码，形式分三种：

①字符和一位像素模式。显示数据全部用来给出光点信息，用CSS给出色调代码（通常称为背景色）。CSS＝0时为绿色，CSS＝1时字符模式下为橙色，图形模式下为黑色。因为光点数据有0和1，产生不同的亮度电平，亮度成分与色调配合，便形成色饱和度的变化，以浅色和深色的反差来显示字符或图形。

②SG4模式。可用DD6～DD4三位表示色调代码，因而可定义如下八种色调。

	绿	黄	蓝	红	浅黄	浅蓝	洋红	橙
DD6	0	0	0	0	1	1	1	1
DD5	0	0	1	1	0	0	1	1
DD4	0	1	0	1	0	1	0	1

③SG6和二位像素图形模式。可用两位数据（SG6是DD7～DD6）定义色调，加上CSS，仍可给出八种色调信息，不过在一种CSS信号下仅能选择其中四种如下：

CSS＝0：　00—绿　01—黄　10—蓝　11—红，

CSS＝1：　00—粉　01—青　10—洋红　11—橙

6. 视频信号合成。

光点信号、色调信号和同步、消隐信号最后合成在一起，并转换为符合电视技术要求的视频模拟信号输出。

视频信号合成电路的核心是彩色编码器。编码器由Y编码、B-Y编码和R-Y编码三个部分组成，是数字逻辑电路，与电视系统的模拟型编码器原理不同。

前面已经讲过，任何色彩信息都可用Y、B-Y和R-Y三个电平信号来界定。这三个物理量不同值的组合就能表示千差万别的各种色彩。例如彩色电视的标准彩条画面，包括三基色(红、绿、蓝)，三补色(青、紫、黄)和中性的白、黑色，便是由8种幅度的U_Y和7种幅度的U_{B-Y}及U_{R-Y}构成。计算机若用同样的数据量也能达到同样的效果。普及型计算机由于对彩色的要求较低，为了简化系统，可用较少的色度参数配色。这样，不但显示的色彩种数较少，色彩的亮度、饱和度也都是固定的。6847八种显示色的配色参数见表4-3(单位V)：

表 4-3

	绿	黄	蓝	红	浅黄	青	洋红	橙
U_Y	0.54	0.42	0.65	0.65	0.42	0.54	0.54	0.54
U_{B-Y}	1.0	1.0	2.0	1.5	1.5	1.5	2.0	1.0
U_{R-Y}	1.0	1.5	1.5	2.0	1.5	1.0	2.0	2.0

注：①亮度信号是负极性，即高电平表示低亮度。

②色差信号可有负值，通过降低参考电平归一为正值表示。

由表4-3可以看出，只需要三种亮度电平和三种色差电平。编码器的作用就是按光点和色彩数据给出相对应的三个数据，再由数字量/模拟量(D-A)转换电路变成所规定的电平输出。

彩色编码器能按不同的模式进行编码。SG4的编码是基本型，三位色调代码除产生两个色差信号外，还同光点信号共同决定亮度信号。光点为亮时亮度信号取上表的值，光点为暗时亮度信号为黑电平(0.72V)。SG6和两位像素模式将CSS作为一位色调代码取入，因为光点信号全为亮，只有一套亮度电平。字符和一位像素模式由CSS决定色差信号，光点决定亮度信号。

编码器的输出经D-A转换后，在B-Y中还要加入色同步选通脉冲，在Y中要加入行同步脉冲和行消隐脉冲。

图4-27是LASER 310输出的彩条画面示意图。它由SG4的"□"变换色彩构成，可用以下程序产生：

```
10  CLS:FOR I=0 TO 15:FOR J=1 TO 8:FOR K=0 TO 8:COLOR J
20  PRINT "□";:NEXT:NEXT:NEXT
30  GOTO  30
```

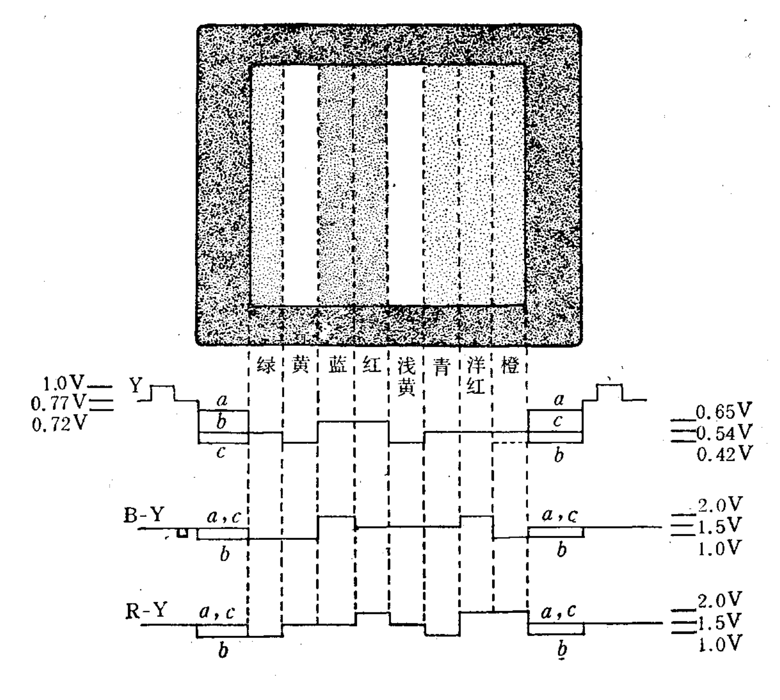


图4-27 彩条信号电平

将彩色电视机的色饱合度旋钮调到最小位置，在屏幕上看到的就是图4-7这样的黑白图像。这时只显示了亮度信号的三个级别，与图下的Y输出电平曲线是一致的。逐渐调大色饱合度，使色差信号参加显示，便能获得八个彩条。

边框部分的电平由模式控制编码器产生。图中不同的边框电平的形成条件是：*a*. $\overline{A}/G=0$(文本模式的黑边框)；*b*. $\overline{A}/G=1$并且CSS = 0(图形模式并且选择绿色背景时，边框与背景相同)；*c*. $\overline{A}/G=1$并且CSS = 1(图形模式并选择浅黄色背景时，边框与背景相同)。由此可以清楚地看出LASER 310不同边框状态的形成机制。上下边框的情形与此相同。

Y的信号为负极性，高电平表示低亮度。边框a电平为0.72V，是图像信号中的黑电平。消隐电平更负于黑电平，为0.77V。行同步脉冲幅度1.0V，所以是不会在屏幕上显示出来的。

(四) 制式调整电路

MC6847是面向NTSC-M制设计的。它的输出信号虽也可调制4.43MHz副载波，用于PAL制的机器，但对照PAL-DK制信号标准，还有一些差别。LASER 310用GA003中的部分逻辑电路来进行制式调整。

首先是行周期。6847行周期63.5μs。LASER 310将它的时钟频率降为3.547MHz后，使行周期变为DK制的64μs。

其次是场频。6847场频60Hz，每场(逐行扫描方式)262行，DK制场频50Hz，每场应为312行。二者相差甚大。LASER 310采取每场插入50行、延长场周期的方法来加以调整。

第三是色同步选通脉冲。NTSC制的色同步信号不携带逐行倒相信息，因此6847只在B-Y输出中含有负的色同步选通脉冲。而按PAL制，则在R-Y输出的同一时刻还应含有一个正

的色同步选通脉冲，以便通过平衡调幅和叠加，产生出逐行倒相的色同步信号。LASER 310在6847外部产生这个脉冲加入R-Y。

制式调整电路框图见图4-28。

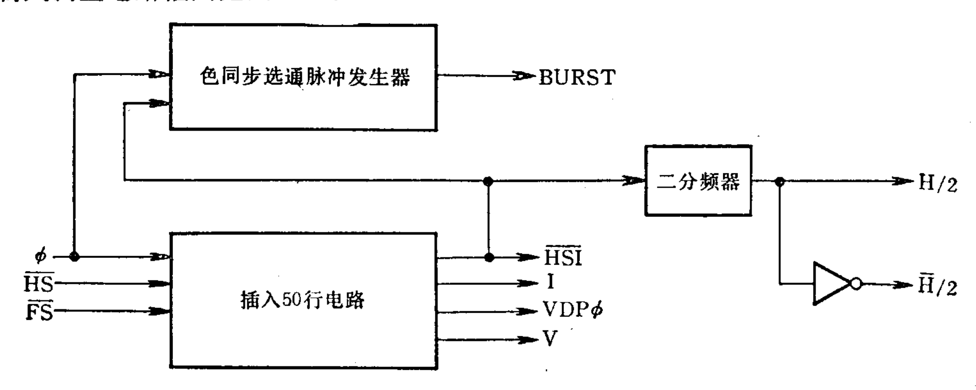


图 4-28　GA003制式调整电路框图

1．插入50行电路。

插入50行电路可由三个部分组成：IC1(双 2 × 5 进异步计数器74LS390)和IC4～IC10各逻辑门组成的时钟切换控制器，IC2(双16进异步计数 器74LS393)和IC11、IC12、IC13组成的附加行同步脉冲发生器，IC3(74LS390)担任的插入50行脉冲计数器。解析图见图4-29。

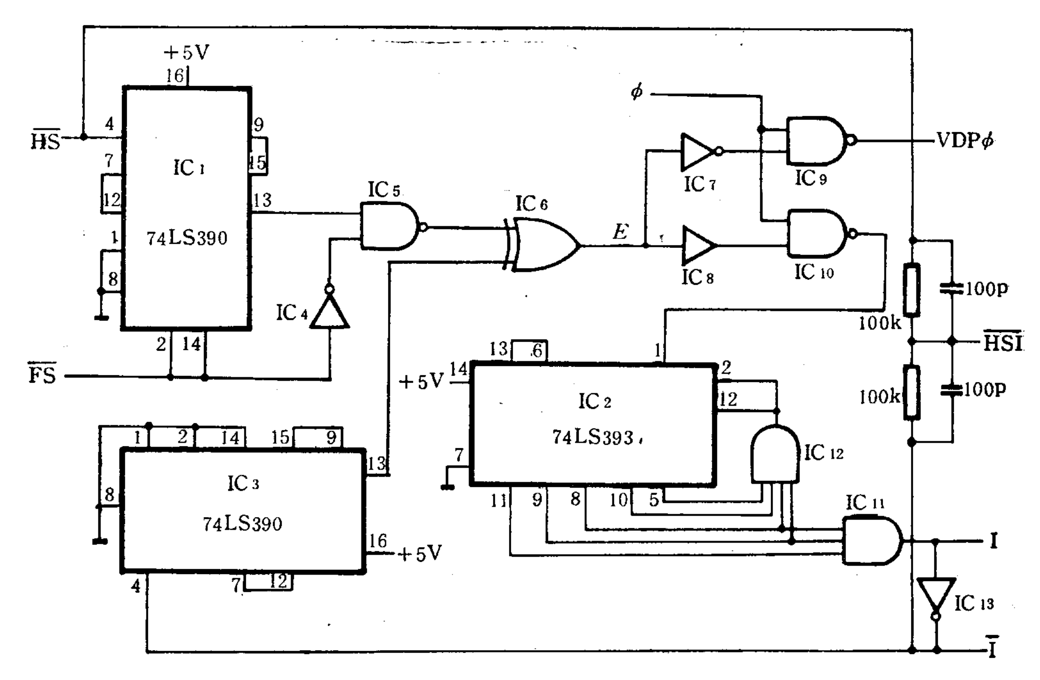


图 4-29　插入50行电路

IC7～IC9构成时钟信号切换电路。GA008产生的 3.54MHz 时 钟 信 号ϕ，经这个电路分配给 6847 或附加行同步发生器使用，由异或门 IC6 输出信号E控 制 切 换。E = 0 时钟脉冲通过IC9送往6847，IC10被封锁，输出恒为高电平。E = 1 时，IC9被封锁，6847时钟“停摆”，

相当于进入一个长延时的时钟周期，各种电平都静止不动。附加行同步发生器则开始工作。

IC2的两个16进计数器串接成256进计数器，对时钟脉冲计数，可并行输出8位二进制计数值（从D7到D0的输出脚排列是8、9、10、11、6、5、4、3）。从计数值为0开始，三输入与门IC11的输出I为低电平。计数到208(11010000B)，8、9、11脚的高电平使I变高。继续计数16个时钟脉冲到224(111000000B)时，I随11脚电平变低。再计数4个脉冲到228(11100100B)，由于四输入与门IC12输出高电平反馈到计数器复位端，使其复位为0，再次循环上述过程。这样，I的周期就是$228\times1/f\approx64\mu s$，正脉冲宽度$16\times1/f$，同6847的行周期和行同步脉冲基本相同。因为这时6847处于静止状态、无同步信号输出，它便可用来参加视频信号合成。

E点的电平是由IC1和IC3这两个计数器共同决定的。它们的一个二进计数器未用，其余部分串接成$(5\times5)\times2=50$进计数器，以二进计数器的数据输出端QA1和QA3(第13脚)为最高位。这种计数器的特性是由正脉冲清零，脉冲下降沿触发计数，计数满25时QA由0变1，再计数25后QA由1变0。

下面分析电路工作的动态过程。

①IC1在6847的垂直同步信号$\overline{FS}$变为有效后，对行同步信号$\overline{HS}$计数，计满25个行脉冲时，$\overline{FS}$尚未升高($\overline{FS}$宽度为32H)，通过IC4、IC5向异或门IC6一端送入0。设此时IC3的QA3也输出0，则E点为0，6847工作暂停，IC2开始产生行脉冲信号I，送往视频信号合成电路。

②IC3对$\overline{I}$计数，计到25个脉冲后QA3变高。因6847停止工作后$\overline{FS}$、$\overline{HS}$都静止，所以IC1输出也未改变，于是E点因IC6输入相异而变高，IC2计数脉冲被封锁，6847恢复工作，IC1继续对$\overline{HS}$计数。

③IC1计数7个行脉冲后，$\overline{FS}$升高无效，IC1被清零，导致向异或门IC6输出1，与QA3的输出相同，E点电平变低，6847再次“停摆”，IC2又产生25个行同步脉冲。然后，QA3变为0，使6847恢复工作。

④在$\overline{FS}$无效期间IC1停止工作，6847时序经过后均衡期、上边框、图像场后，$\overline{FS}$再度有效。于是回到第①步。循环进行，直至关机。

经过这样的处理，使每场增加了50行。对照6847的垂直时序，正好在场同步脉冲的两侧各插入25行。在产生50个行同步脉冲的同时，6847的Y、R-Y和B-Y输出均维持在工作中断时的状态，即边框和消隐电平，无需另外添加。插入的50行周期共3.2ms，使CPU可占用VRAM地址线的时间延长到7ms。

$\overline{I}$同$\overline{HS}$合并在一起，便成为完整的行同步信号$\overline{HSI}$，可用来控制色同步选通脉冲和PAL开关信号的生成。

2. 色同步选通脉冲的生成。

色同步选通脉冲发生器由二单稳态多谐振荡器(74LS123)担任，解析图见图4-30。两个振荡器都接成脉冲上升沿触发方式。振荡器M1由$\overline{HS}$信号触发，其输出的负脉冲上升沿触发振荡器M2，实际只起延时电路作用。振荡器输出脉冲的宽度由外接RC网络的时间常数决定$t_w=0.45R_r\cdot C_r$。

当行同步信号$\overline{HS}$变高时，经两级反相器缓冲延时后触发M1，使Q1输出电平降低。经$0.45\times1.8\times10^3\times0.001\times10^{-6}=0.81\mu s$，加上输入的延时，距$\overline{HS}$上升沿约1μs。Q1的上升沿触发了M2，由Q2端输出一个正脉冲，宽度为$0.45\times18\times10^3\times390\times10^{-12}=3.2\mu s$。这个正脉冲与

的色同步选通脉冲，以便通过平衡调幅和叠加，产生出逐行倒相的色同步信号。LASER 310在6847外部产生这个脉冲加入R-Y。

制式调整电路框图见图4-28。

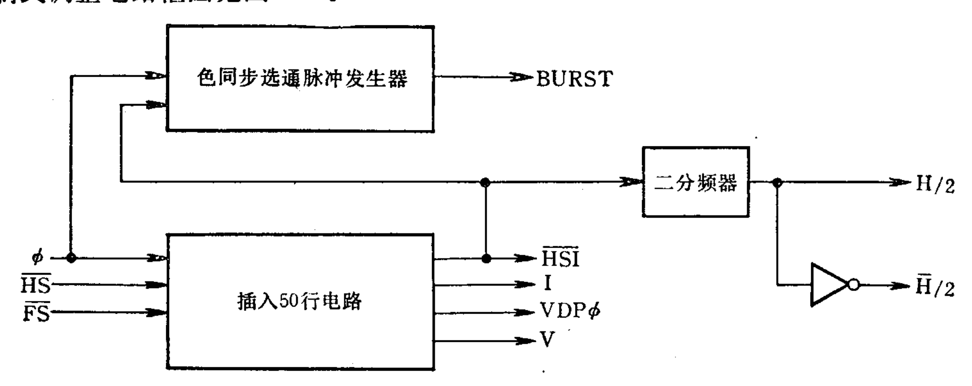


图4-28 GA003制式调整电路框图

1. 插入50行电路。

插入50行电路可由三个部分组成：IC1(双2×5进异步计数器74LS390)和IC4～IC10各逻辑门组成的时钟切换控制器，IC2(双16进异步计数器74LS393)和IC11、IC12、IC13组成的附加行同步脉冲发生器，IC3(74LS390)担任的插入50行脉冲计数器。解析图见图4-29。

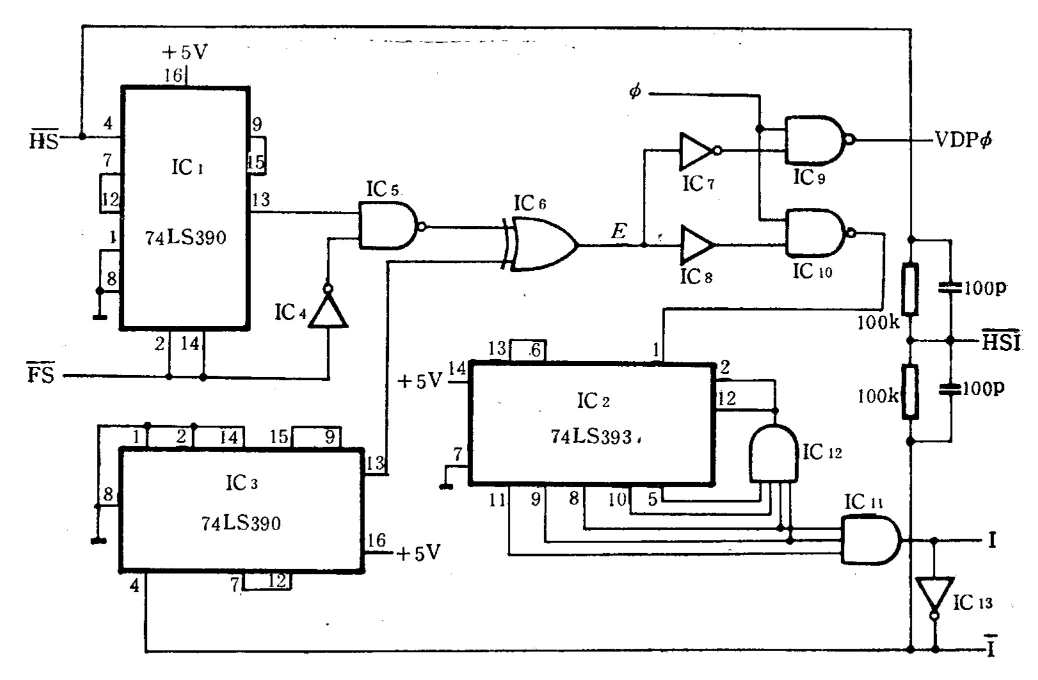


图4-29 插入50行电路

IC7～IC9构成时钟信号切换电路。GA008产生的3.54MHz时钟信号ϕ，经这个电路分配给6847或附加行同步发生器使用，由异或门IC6输出信号E控制切换。E＝0时钟脉冲通过IC9送往6847，IC10被封锁，输出恒为高电平。E＝1时，IC9被封锁，6847时钟“停摆”，

相当于进入一个长延时的时钟周期，各种电平都静止不动。附加行同步发生器则开始工作。

IC2的两个16进计数器串接成256进计数器，对时钟脉冲计数，可并行输出8位二进制计数值（从D7到D0的输出脚排列是8、9、10、11、6、5、4、3）。从计数值为0开始，三输入与门IC11的输出I为低电平。计数到208(11010000B)，8、9、11脚的高电平使I变高。继续计数16个时钟脉冲到224(111000000B)时，I随11脚电平变低。再计数4个脉冲到228(11100100B)，由于四输入与门IC12输出高电平反馈到计数器复位端，使其复位为0，再次循环上述过程。这样，I的周期就是$228\times1/f\approx64\mu s$，正脉冲宽度$16\times1/f$，同6847的行周期和行同步脉冲基本相同。因为这时6847处于静止状态、无同步信号输出，它便可用来参加视频信号合成。

E点的电平是由IC1和IC3这两个计数器共同决定的。它们的一个二进计数器未用，其余部分串接成$(5\times5)\times2=50$进计数器，以二进计数器的数据输出端QA1和QA3(第13脚)为最高位。这种计数器的特性是由正脉冲清零，脉冲下降沿触发计数，计数满25时QA由0变1，再计数25后QA由1变0。

下面分析电路工作的动态过程。

①IC1在6847的垂直同步信号$\overline{FS}$变为有效后，对行同步信号$\overline{HS}$计数，计满25个行脉冲时，$\overline{FS}$尚未升高($\overline{FS}$宽度为32H)，通过IC4、IC5向异或门IC6一端送入0。设此时IC3的QA3也输出0，则E点为0，6847工作暂停，IC2开始产生行脉冲信号I，送往视频信号合成电路。

②IC3对$\overline{I}$计数，计到25个脉冲后QA3变高。因6847停止工作后$\overline{FS}$、$\overline{HS}$都静止，所以IC1输出也未改变，于是E点因IC6输入相异而变高，IC2计数脉冲被封锁，6847恢复工作，IC1继续对$\overline{HS}$计数。

③IC1计数7个行脉冲后，$\overline{FS}$升高无效，IC1被清零，导致向异或门IC6输出1，与QA3的输出相同，E点电平变低，6847再次“停摆”，IC2又产生25个行同步脉冲。然后，QA3变为0，使6847恢复工作。

④在$\overline{FS}$无效期间IC1停止工作，6847时序经过后均衡期、上边框、图像场后，$\overline{FS}$再度有效。于是回到第①步。循环进行，直至关机。

经过这样的处理，使每场增加了50行。对照6847的垂直时序，正好在场同步脉冲的两侧各插入25行。在产生50个行同步脉冲的同时，6847的Y、R-Y和B-Y输出均维持在工作中断时的状态，即边框和消隐电平，无需另外添加。插入的50行周期共3.2ms，使CPU可占用VRAM地址线的时间延长到7ms。

$\overline{I}$同$\overline{HS}$合并在一起，便成为完整的行同步信号$\overline{HSI}$，可用来控制色同步选通脉冲和PAL开关信号的生成。

2. 色同步选通脉冲的生成。

色同步选通脉冲发生器由二单稳态多谐振荡器(74LS123)担任，解析图见图4-30。两个振荡器都接成脉冲上升沿触发方式。振荡器M1由$\overline{HS}$信号触发，其输出的负脉冲上升沿触发振荡器M2，实际只起延时电路作用。振荡器输出脉冲的宽度由外接RC网络的时间常数决定$t_w=0.45R_r\cdot C_r$。

当行同步信号$\overline{HS}$变高时，经两级反相器缓冲延时后触发M1，使Q1输出电平降低。经$0.45\times1.8\times10^3\times0.001\times10^{-6}=0.81\mu s$，加上输入的延时，距$\overline{HS}$上升沿约1μs。Q1的上升沿触发了M2，由Q2端输出一个正脉冲，宽度为$0.45\times18\times10^3\times390\times10^{-12}=3.2\mu s$。这个正脉冲与

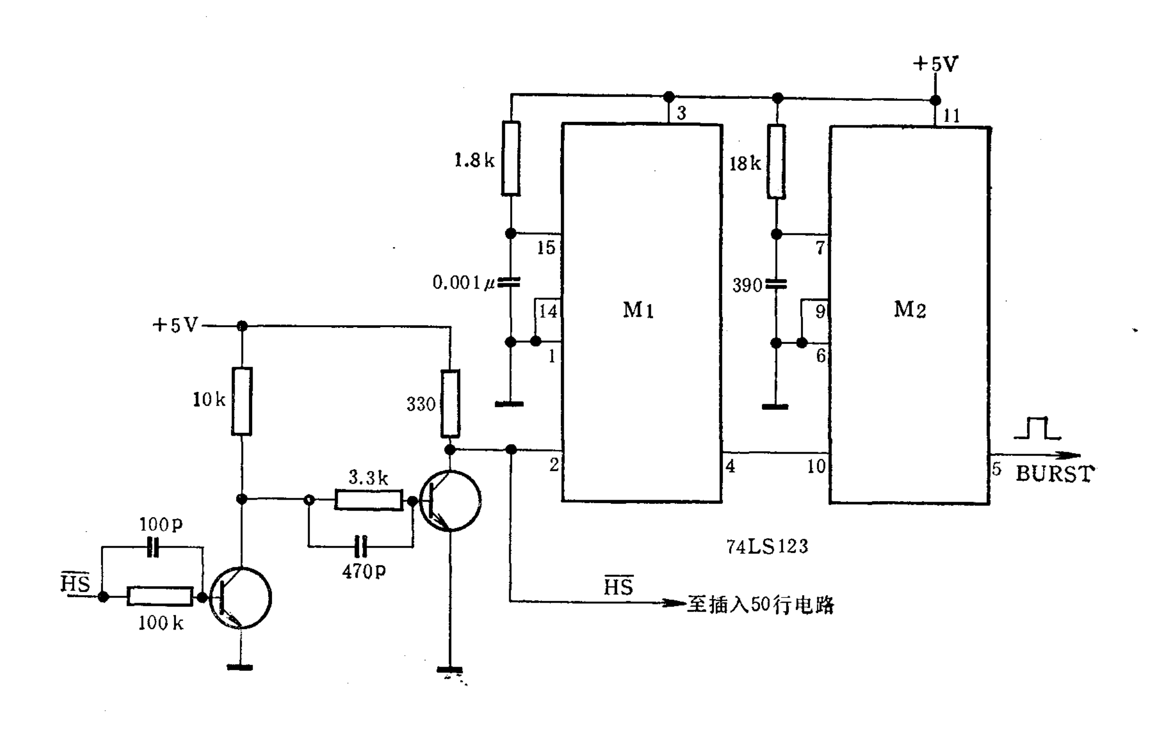


图 4-30 色同步选通脉冲发生器

6847的色同步选通脉冲时序相同而极性相反，符合PAL制要求。

（五） 后端电路

1。彩色调制器。

LASER 310采用PAL制彩色调制集成电路TBA520，其结构框图参见4-17。它包含两个平衡调幅器和加法电路，还带有PAL开关。

如图4-31所示，6847的色差信号R-Y送入调制器之前先混入正的色同步选通脉冲，同B-Y中已有的负脉冲配对。两路色差信号输入端都加有旁路电位器，可通过调节它们改变输入电平比值从而微调色调。

从GA008分频器送来的4.43MHz副载波，经过移相电路变成相位相差90°的两个副载波。其中一路需经PAL开关逐行倒相。PAL开关由1/2行频脉冲控制。这个开关信号由$\overline{HSI}$经二分频形成。

两个色差信号经调制后叠加成PAL色度信号，并且含有逐行倒相的色同步信号。它由TBA520第5脚输出。

2。视频信号合成电路。

视频信号合成电路见图4-32。它是一个晶体管加法电路。6847的亮度信号Y送入基极，色度信号F送入发射极。因Y中不含插入的50个行同步脉冲，故还要由集电极送入I。通过晶体管的非线性作用，三者重合为FBAS信号，送到机器的MONITOR输出插口。

3。射频调制器。

为了用普通电视机作显示设备，普及机一般都具有射频调制器。LASER 310配置的射

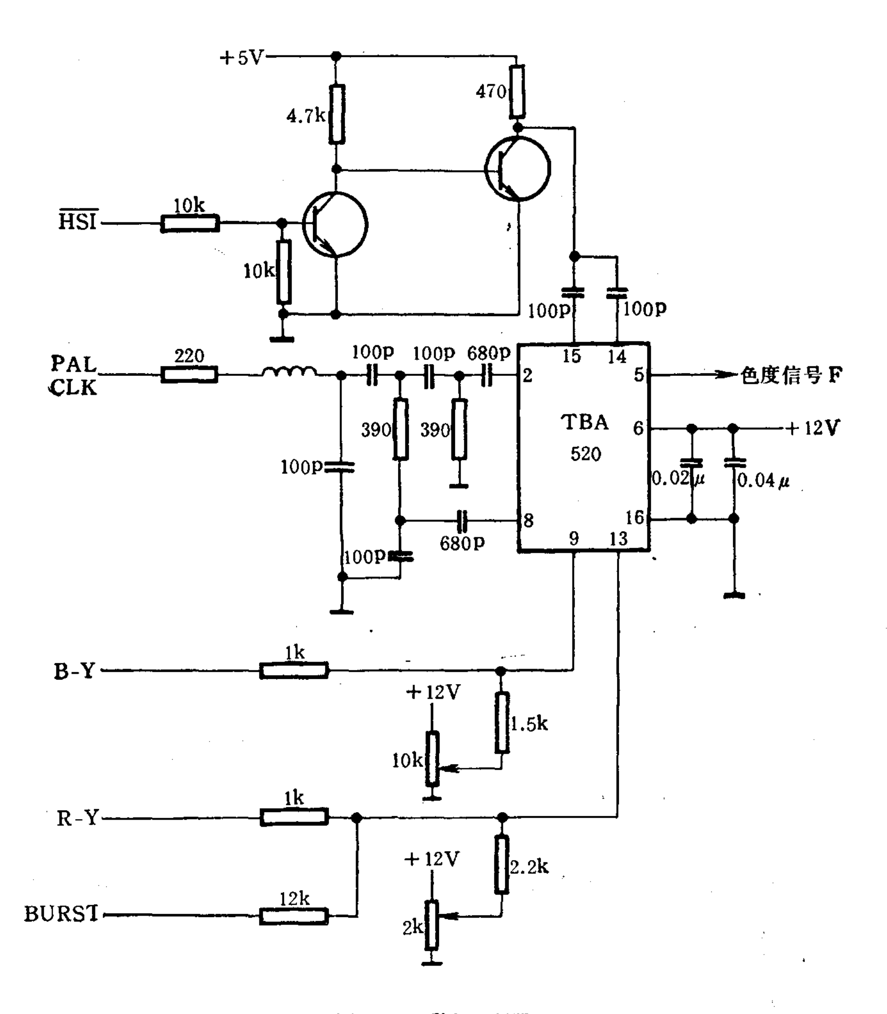


图 4-31 彩色调制器

频调制器有超高频段(UHF)和甚高频段(VHF)两类。国内引进的多为VHF型(一般是二频道,图像载频57.75MHz),以便老式的十二频道黑白电视机也能接收。

射频调制器的核心是一个高频振荡器。振荡频率由LC网络决定。用视频信号控制基极电位,振荡的幅度便随视频信号的变化而变化,成为射频的调幅波。它虽能向空间发射,但功率很小,还不足以供电视机天线接收。一般都用同轴电缆闭路传送。

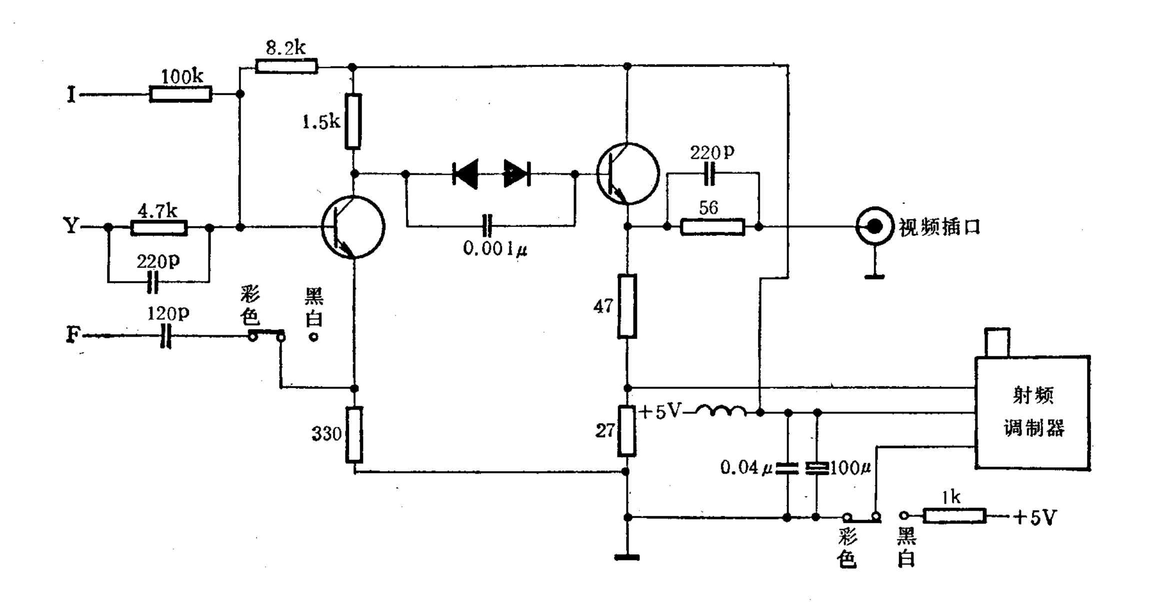


图 4-32 视频信号合成电路

第五章　Z80机器语言简介

计算机以CPU执行预先存贮在存贮器里的程序的方式来工作。这种程序由CPU能够“认识”的数码化的机器指令组成，称为“机器语言程序”。各种CPU间，除从设计上就考虑了兼容者（如Z80同8080）外，相互“语言不通”，各有自己的指令系统。

计算机的系统程序一般都是机器语言程序。LASER　310操作系统——V2.0便是用Z80机器语言编制而成。要对它进行剖析，当然必须懂得机器的语言。为了使每个读者都能读懂它，这一章从实用的角度简要介绍Z80指令系统以及汇编语言的基本常识。更系统完整的知识请读者阅读有关专门著作。

一、机器语言和汇编语言

（一）　机器语言

1．机器字。

机器语言程序是机器字的有序集合。字长8位的机器，每个机器字是一个字节的二进制数码（一般也称为机器码，书写时常用两位十六进制数来表示），其值由00H～FFH，只有256种。机器字同汉字一样也有一字多义的特点，同一个字在机器语言中可能有多种涵义。例如C1H这个机器码，就可能是：一条指令，C1——将堆栈顶部两字节数据弹出，装入BC寄存器对；一条指令的操作码的一部分，如CBC1——将寄存器B的D0置为1；一个参加操作的数，如1EC1——将数C1H装入寄存器B；一个操作数的一部分，如21C101——将数01C1H装入HL寄存器对；一个I/O口地址，如D3C1——把A中的数送到地址为C1H的I/O口；一个内存单元地址的一部分（高八位或低八位），如22AAC1——把HL寄存器对的内容装入地址为C1AAH和C1ABH二单元；一个地址偏移量，如18C1——跳转到与PC中当前地址相距C1H个单元的地址去执行。

那么，怎样确定一个机器字的涵义呢？也同汉语相仿，要看“上下文”——也就是说要由它在程序中的地位决定，详见后述。

2．机器指令和程序。

机器语言程序由一条条指令组成。一条指令由一个或几个机器码组成。指令的作用类似于BASIC的语句，能使CPU进行一项基本操作。

一条指令一般包括“操作码”和“操作数地址码”两部分。

一个操作码不仅决定CPU进行何种操作，也决定如何取得操作数——寻址方式，同时还决定存放结果的目标地址。操作数地址码则是在前者规定的寻址方式下，指出操作数所在的地址（寄存器、内存单元或I/O口）以供CPU取来参加操作。在立即寻址方式下，操作码后面就是操作数本身（这称为“立即数”，其地址就是当前的PC指针）。

例如，21　E9　7A和　22　E9　7A这两条指令，都是把一个16位二进制数传送给HL寄

存器对。但前者的操作码21H为“(扩展)立即寻址”方式，CPU将连续两次将PC内容送上地址总线进行读操作，从操作码后面的两个单元取操作数，这个数便是7AE9H。而操作码22H的寻址方式为“扩展寻址”，CPU同样把操作码后面两个单元的内容(7AE9H)取入，但并不装入HL寄存器，而是作为一个内存地址，分两次(7AE9H和7AEAH)送上地址总线进行读操作，再将取得的数据送进HL。显然，二者执行结果的HL寄存器内容是完全不同的。

一条指令的构成有以下几种形式：

(1)少数指令没有操作数，如F3H(关中断)等CPU控制指令。

(2)以一个字节中的几位为操作码，另几位为地址码(仅限于表示寄存器)，把二者“压缩”在同一个机器字中，成为单字节指令。例如(下划线的为地址码)00000100B(04H)——把B的内容加1；00010100(14H)——把D的内容加1。

(3)第一个字节是操作码，第二个字节是地址码，构成双字节指令，如FE41(将A中的数据与操作数后面单元中的立即数41H比较)。

(4)第一个字节是操作码，后跟两个字节地址码，构成三字节指令，如C31A19(转移到191AH单元的指令去继续执行)。

(5)凡以CBH，DDH，EDH或FDH开头的，均与第二个字节组成双字节操作码，地址码可为隐含、一字节、两字节三种，分别构成双字节、三字节和四字节指令，如CB4F(测试A的D_1位)，DD7E05(将IX＋05H的地址中的数据装入A)，DD210080(将8000H装入IX作为变址寻址的基地址)，等等。

机器语言程序没有BASIC那样的行号、分隔符、结尾符等标识，完全像一篇不加标点的中国文言文，常常使初学者望而生畏。那么，CPU是怎样进行“断句”从而读懂它的呢？

第一，CPU复位或转移到新的程序段起始地址时，第一个M1周期取得的必然是操作码或单字节指令。

第二，操作码如果是CBH，DDH，EDH或FDH之一，下一单元则是操作码的第二部分，CPU将用又一个M1周期将它取出再进行译码，否则，第二字节便是操作数地址码。

第三，对于每一个操作码，其操作数地址码的有无或字节数都是一定的。经M1周期取入操作码并经译码后，便决定了后继的微操作和机器周期数，准确地取出操作数。

第四，地址码取完后，PC正好指向下一个操作码，于是回到第一步。

如此循环进行，程序便得到正确地执行。我们阅读机器语言程序，也要模拟这种方式。

由上述可知，一个机器字的具体涵义和作用是由它在程序中的位置来决定的，所以机器语言程序不容许错码和错位。多了、少了或错了一个码，或转移目标的第一个单元不是操作码，不但影响本条指令，而且会使CPU把下面的操作码和地址码弄颠倒，引起“连锁反应”，造成程序“迷路”，面目全非。

3. 数。

Z80机器指令中的操作数，只有八位和十六位二进制数两种，范围分别为00H～FFH和0000H～FFFFH。作为无符号的整数，可分别表示0～255和0～65535。要特别说明的是，十六位二进制数在内存中是存放在两个连续单元中的，地址较高的单元存放高八位，地址较低的单元存放低八位，该数则以低字节单元的地址为其地址。所以按照内存地址的顺序，是低字节在前，高字节在后。如78A4H这个数，在内存的排列却是A4 78。但在文字叙述和汇编语言中

仍按日常的数学表达习惯，高字节在前，低字节在后。

单字节和双字节数，都可以作为有符号数。最高位为0时作正数，最高位为1时作负数。这样它的绝对值便只有7位和15位了，数值范围分别为：-128～+127和-32768～+32767。在相对跳转和变址寻址的指令中，CPU自动将单字节的立即数作补码形式的有符号数处理（正向或负向跳转、寻址）。

参加运算的数是有符号数还是无符号数，是否是小数，小数点在何处，以及多字节数等等，都由程序员自行设定、处理和识别，CPU无此能力。

4．寄存器、标志和堆栈的使用。

Z80的大部分指令都要使用各通用寄存器。在机器语言程序中，寄存器的使用非常频繁，常有不够用的情况。程序员必须随时记住它们的内容和变化，否则容易出错。

状态标志也是学习机器语言程序的一个要点。必须了解每条指令对每个标志位的影响，并随时跟踪它们的变化。执行条件转移、条件调用、条件返回和带C标志的运算时，所依据的标志状态有时是在很多条指令之前就形成的。

使用堆栈的要领是必须牢记栈中的内容和存入的顺序，严格遵循“后进先出”原则，不能弄错。CPU也要用堆栈保存CALL、RST指令的返回地址。如存取顺序错乱，把返回地址当成所存数据取出，不但会造成运算错误，而且到应返回时，CPU必将把另外的数据当成返回地址，问题就更大了。

（二） 汇编和反汇编

直接用机器码编制的程序，不但难懂难记，而且易错难纠。所以微处理器的生产者为用户提供了一套与机器指令对应的符号指令。例如，机器指令80H，很难看出是什么意思，相应的符号指令是 ADD A,B(将A的内容和B的内容相加，结果放入A)，意义一目了然。

当然，CPU并不能执行符号指令。用符号指令写成的程序，要先用一种“汇编程序”软件把它翻译成机器语言程序。这种翻译过程，叫做“汇编”。因而，用符号指令写成的程序就叫“汇编语言程序”。同样，机器语言程序也可以翻译成汇编语言程序，这叫“反汇编”。LASER 310本身没有配置汇编程序。这里向读者推荐郑州张涛开发的《LASER 310 DEBUG》软件。它具有较强的汇编和反汇编功能。没有汇编程序帮助，也可自己通过查表进行手工汇编或反汇编，虽然比较费功夫，但对学习汇编语言很有助益。

编制汇编语言源程序时必须注意，Z80指令系统的符号指令是固定和有限的，不能想当然地用合法的符号组成非法的“指令”。如 ADD A，B是Z80的符号指令，而 ADD B,C就不是了，因为它没有对应的机器指令，CPU也不具备这样的操作。

一条符号指令的一般构成是： 操作助记符 操作数表示符

1．操作助记符。

是一个表示操作性质的英文单词或其缩写，如 CALL,LD(LOAD)等。操作助记符并不完全对应于操作码，因为一些操作同类型而寻址方式和目标地址不同的指令使用同一操作助记符。

2．操作数表示符。

用以表示寻址方式、操作数地址和目标地址。

地址为寄存器时，直接用寄存器名表示。地址为内存某单元时，用双字节数、寄存器对或变

址值指出，I/O口地址用单字节数，寄存器C指出，均须括在括号内，如(78F9H)，(DE)，(IX+10)(FF)，(C)等等。无括号的数表示立即数。

除(SP)指堆栈顶端的二单元外，用寄存器对或变址值只可指定一个存贮单元。用十六位内存地址，可指定一个单元或两个连续单元。怎样区分呢？那就要看参加操作的是单个寄存器还是寄存器对。例如：

LD　A,(7AE9)

LD　HL,(7AE9)

INC (HL)

第一条是八位数据传送，因为A是八位寄存器：A←(7AE9)。

第二条是十六位数据传送，因为HL是十六位寄存器对：L←(7AE9)；H←(7AEA)。

第三条是八位数据加一，因为(HL)只能指定一个存贮单元：(7AE9)←(7AE9)+1。

操作数表示符中有两个元素(地址或立即数)时，用逗号分隔，逗号前面的一个是存贮操作结果的目标地址，在算术逻辑运算中并存有操作数之一。

在概括地说明指令时，我们用特定符号代表某一类型的数：

N　表示八位二进制数(单字节，两位十六进制数)。

NN 表示十六位二进制数(双字节，四位十六进制数)。

DJS　表示相对跳转类指令中的八位有符号数(位移量)，有时亦用"*e*"表示。

IND　表示变址寻址类指令中的八位有符号数(位移量)。

b　表示位操作指令中的目标位(0～7)。

此外，用cc表示指令要依据的条件标志状态，种类和意义是：

Z　当Z标志为1时；

NZ 当Z标志为0时；

C　当C标志为1时；

NC 当C标志为0时；

P　当S标志为0时；

M　当S标志为1时；

PO 当P/V标志为0时；

PE 当P/V标志为1时。

不需要操作数的指令，仅有操作助记符。有一些八位二进制数的算术、逻辑运算指令，只给出一个操作数表示符，第一个操作数表示符隐含为A，如　OR　B(A←A OR B)。

汇编语言程序的格式一般是：

标号：　符号指令　；注释

其中"标号"是一个字符串，用作指令(一般也是程序段的入口)或数据的标识，供汇编程序借以找到它。这样，调用或转入时就不必具体指明其地址，而可用标号代替，实际的物理地址由汇编程序通过搜索找出后自行产生地址码。可见，并不是每条指令都需要加上标号。"注释"段仅供编程者自己和程序读者参考，汇编程序不予处理。

此外，汇编语言程序还可使用若干"伪指令"，用来指出程序的起始地址和结束，给标号赋值，划定数据存贮空间等，只供汇编程序在汇编时使用，并不形成机器码。

为了便于初学者阅读，本书所载的汇编语言程序文本采取以下格式：

指令的首地址- 指令的机器码序列 符号指令 ；指令的汉字解说。

不用标号和伪指令，地址和符号指令中的数都用十六进制，数制标志“H”一律省略。

二、 Z80的符号指令

Z80指令系统是8位MPU中最为丰富的，共有158种、近700条不同的指令。下面只从实用的角度对初学者作简要解释，以助记符为出发点讲述。

（一） 传送指令

助记符 LD(装入、传送)

传送指令的功能类似BASIC的LET语句，它的操作是把操作数从源地址送到目标地址。执行结果，目标地址内容变得与源地址相同，而源地址内容不变。

传送指令包括八位和十六位二进制数两类，区别见上节。以LD为助记符的指令，除LD-A,I和LD A,R外，对标志寄存器各位均无影响。

（二） 交换指令

助记符 EX(交换)；EXX(主、辅寄存器交换)

交换指令是使指定的寄存器对之间(DE同HL，AF同AF′)，或寄存器对(HL,IX,IY)同堆栈栈顶两单元(地址为(SP))交换内容。

后一种情况，例如 EX (SP),HL 执行后，堆栈顶上的两字节内容存入HL，这两个单元的内容被HL原有内容取代，栈顶地址和堆栈指示器SP的值均未改变。

EXX是Z80一条功能很强的指令，能够一次把BC,DE,HL三对主寄存器和BC′,DE′,HL′的内容互相交换，使主寄存器内容方便地保存起来，再用一次EXX，即可全部取回。此指令常与EX AF,AF′联用，在响应中断时用来保护现场。

交换指令对各标志均无影响。

（三） 堆栈操作指令

助记符 PUSH(压入)；POP(弹出)

Z80的堆栈原则上可以建立在用户RAM的任何地方，规模只受RAM低端的限制。堆栈的栈底(最高地址)可用 LD SP,HL指令设定。

堆栈操作指令适用于各主寄存器对、AF和变址寄存器。PUSH的功能是将源寄存器对的内容存入当前栈顶以上两单元(高字节在下)；POP是将堆栈顶端两单元内容取出，送入目标寄存器对。两种操作的同时堆栈指示器SP都将自动－2或＋2，指向新的栈顶。PUSH操作后，源寄存器的数据并未丢失，仍可使用。POP后，原单元内容虽然也未改变，但因已被“排出”堆栈之外，用堆栈操作指令不能再取用。PUSH同POP应两两配对，并牢记存取的顺序。当程序分支较多时，在这个问题上最易出错。同一组数据，压入和弹出的寄存器对完全可以不同，有时就利用堆栈在寄存器对间传送数据，例如：

```
PUSH DE
POP  BC
```

把DE的内容送入BC。堆栈一旦有了已经无用的数据，应及时用闲置的寄存器将它们取掉。有必要从非栈顶位置取出数据时，可先用寄存器暂时保存其上面的数据，待目标数据弹出

后再依原来顺序压入。

Z80没有单字节的栈操作指令，要保存单个寄存器内容也只好连配对的另一个一起存栈。甚至为了保存标志寄存器 F 中的某一位，以备将来用作转移的依据，也得用 PUSH AF 指令一次存入十六位数据。

堆栈操作指令对各标志位均无影响。

（四） 算术运算指令

1．加法和减法指令。

助记符 ADD(加)；ADC(带进位标志加)；SUB(减)；SBC(带进位标志减)

算术指令分别为八位和十六位操作。八位加减指令中用来存放被加、被减数并存放结果的必是累加器A，符号指令中只给出加数和减数(或地址)。十六位加减操作只能在规定的寄存器对之间进行。

如前所述，由于寄存器位数的限制，C标志实际起了"第 9 位"或"第17 位"的作用，可存放进位或借位的 1，无进、借位时C标志复 0。所以，只看目标寄存器本身，有时就会出现"越加越小，越减越大"的"怪事"。如当A的值为 0 时，执行 SUB 01H指令后，A 中成为FFH (C＝1)。同理，当HL＝8000H，DE＝FFCEH时，ADD HL，DE的结果，HL 变成 7 FCEH (C＝1)。运算后可用C标志来判断结果和进行下一步运算。

带进位的加减指令就是为此而设的。它除将两个操作数相加减外，还要加上或减去C标志的值。对多字节数作加减运算时，操作数是存放在内存单元中的，每次取出一或二字节到寄存器中，进行对位运算。从最低字节开始时，用不带进位的算术指令，往高字节进行时都用带进位(借位)的指令，后面字节运算时的进位和借位就都算进去了。这同笔算方法相似。十六位减法没有不带借位的操作指令，所以，不需要减C标志值时，一定要先将C标志复位为 0，以免有时多减 1 造成错误结果。

加减指令将影响各标志位，只有十六位的ADD指令对Z、S和P/V标志无影响。

2．比较指令。

助记符 CP(比较)

比较也是八位的减法运算，只是减的结果不存，A的内容不变，其余与SUB相同。执行后可从标志中看出操作数与A值比较的结果：Z(＝A)；NZ，C(＞A)；NZ，NC(＜A)。

3．加一、减一指令。

助记符 INC(加 1)；DEC(减 1)

这类指令可用来使指定的寄存器或寄存器对内容加 1 或减 1，通常用作记数。也可将一个内存单元的内容加 1 减 1，单元地址须用HL或变址寻址方式给出。

八位加 1 减 1 指令将影响除 C 标志之外的各标志位。十六位此类指令则不影响任何标志。这是它与一般加减指令明显不同之处，不能靠它来产生转移条件。

4．逻辑运算指令。

助记符 AND(与)；OR(或)；XOR(异或)

逻辑运算均为八位操作，并不是在两个八位二进制数间进行，而只是位与位之间的运算，在累加器A同另一操作数的八个位之间同时、分别独立进行，结果存于A，另一操作数不变。运算规律可归纳为四句口诀："双方皆 1，AND得 1；任一为 1，OR得 1；双方不同，XOR 得 1；

除此之外，一概得0。”

逻辑运算影响各标志位。它们在程序中用得较多，但并不是真的用来“运算”，常用它们的功能达到以下目的：

（1）将C标志清零。逻辑运算不可能进位，必定使C标志复位为0。

（2）累加器A清零。用XOR A指令，让A与A本身作异或运算，因各位完全相同，结果八位必然全部成为0。

（3）测试累加器A内容是否为0。用OR A指令，让其本身作或运算，只有当A为0时，结果才会为0，由Z标志可以得知（亦可用AND指令测试，道理相似）。执行后A内容不变。

（4）测试某寄存器或以(HL)指出的内存单元是否为0。先将A清零，再与它作OR运算，只有该操作数为0时结果方为0。

（5）测试一个寄存器对或连续的两个内存单元的内容是否为0（如程序结束标志）。将一个单元内容取入A，再与另一单元内容作OR运算，二者皆为0时结果方为0。

（6）使一个字节中一些指定的位变成0（屏蔽），而不影响其他位。用一个操作数作为“模板”，它的各位与被处理数相对应，要屏蔽的位为0，要保留的位为1。二者作AND运算即可。例如，有数11010101B，只需要保留D3～D6（下划线部分），则可用01111000B与其作AND运算，结果必为01010000B，达到目的。

（7）使一个字节中指定的位变反，而保持其余位不变。用一“模板”字节，对应于取反的各位为1，其他位为0，同目标字节作XOR运算即可。

5．转移指令。

助记符 JP（转移）；JR（相对跳转）；DJNZ（循环跳转）

无条件转移指令（JP NN）的作用类似BASIC的GOTO语句。条件转移指令（JP cc NN）则类似于BASIC的IF…THEN语句。如当时指定的条件标志状态与指令中的cc一致，则转移，否则按原顺序执行下条指令。转移指令的操作数显然是一个内存地址，为什么不用括号呢？因为从CPU的操作来看，转移指令实质上是以程序记数器PC为目标地址的传送指令，这儿的地址值，只是要送入PC的数据，所以取立即数的形式。

JR指令的操作数不是一个十六位的地址值，而是一个八位的位移量——转移目标地址与当前地址相距的字节数。它实质上也是对PC的操作，但不是传送，而是相加（PC←PC+DJS）。它与JP相比，不仅指令长度减少一字节，而且可以不用了解目标指令的绝对地址，整个程序在内存的地址变更并不影响它的正确跳转。关于JR指令有两条重要的规则：

第一，计算跳转的基地址不是JR指令所在的地址，而是它下一条指令的地址（因为CPU取JR指令后，PC已增值指向此地址）。

第二，指令中给出的位移量是单字节有符号数（补码），00H～7FH为正（向高地址）跳转，80H～FFH为负（向低地址）跳转。所以跳转范围仅可在-128～+127字节，超出此范围不能用JR指令。

在汇编语言程序中，相对跳转指令中的地址码可用标号或绝对的目标地址，写成JR NN，JR cc NN的形式，由汇编程序去计算位移量装入目标程序。手工汇编和反汇编时计算公式是：目标地址-当前地址=位移量。 当前地址+位移量=目标地址。

作这种运算的要领，是正确地处理八位补码与等值的十六位补码之间的转化。若前者化

为后者要增加高八位，使D15～D8每位都等于D7（原符号位），即D7 = 0时，高八位为00000000B(00H)，D7 = 1时，高八位为11111111B(FFH)。例如12H(+18)扩展成0012H(+18)，E1H(-31)扩展成FFE1H(-31)，值均未变。同样，当一个十六位有符号数的D15～D7各位均相同时，亦可舍弃高八位，“压缩”为等值的八位有符号数。如不符合这一条件，说明它的绝对值已超出7位的范围，不能转化。

上面第一道算式的结果，必须在八位有符号数范围内。如在1220H有一条JR指令，执行后的当前地址为1222H，目标地址是1208H。DJS = 1208H - 1222H = FFE6H(借位不计) = E6H，指令汇编成机器码就是18E6。但如欲以1000H为目标地址，计算结果是FDDEH，就在跳转能力以外，不可能汇编了。

第二个算式要先将位移量化成十六位有符号数，计算时最高位向前的进位可略去不计。仍以上面的指令为例：

1222H + FFE6H = 1208H(进位不计)。

DJNZ指令的作用类似于BASIC的FOR…NEXT语句，用以反复执行同一程序段。它以寄存器B的内容为循环变量，初值应在程序中先行赋给，执行DJNZ指令时，先将B减1，如不为0则跳转到目标地址再次执行，直到B减1等于0时，循环结束，顺序往下执行。可以把DJNZ视为特殊的JR指令，其余规定二者相同。循环中要注意保护B的内容不被破坏。循环次数最多256次(B的初值为00H时，第一次减1便成了FFH)。

所有转移、跳转指令(包括DJNZ)对各标志均无影响。

6. 调用、返回指令。

助记符　CALL(调用)；RET(返回)；RST(零页调用)

CALL类似于BASIC的GOSUB，RET类似于BASIC的RETURN，分别用来控制程序转入子程序和返回主程序。

CALL与JP的不同之处，是执行时要先将自己下一条指令的地址(断点)压入堆栈，再执行与JP相同的操作。执行RET指令时，堆栈顶部的两个字节(应为返回地址)弹出，装入PC，于是转到此地址执行。它们都分为无条件和有条件两类，条件标志表示法同JP类指令。

RST又称作“重新启动”指令。它的功能与CALL相同，不同的是它们只能调用“零页”(指0000H～00FFH)内八个固定地址的子程序入口，也用RET返回。这类指令多用于调用中断服务子程序。因它们具有单字节和快速度的优点，当中断系统不使用时，可用作常用子程序入口。

一般说来，调用与返回指令须互相匹配，成对使用。并应保证执行返回指令前堆栈顶部正好是原来的断点地址值，但也有例外，如使用条件返回指令，对于一个CALL或RST，可有多个返回分支。为了将来由子程序返回特定的地址(而不是当前地址)，可用PUSH指令将该地址压入堆栈，再用转移指令转入子程序。这样便只有RET而无CALL。进入子程序后，因某种原因不需要返回原断点，可用POP指令将返回地址清除，这就有CALL而无RET了。

本类指令不影响任何标志。

7. 位操作指令。

助记符　BIT(位测试)；SET(置位)；RES(复位)

位操作指令用来对各个主寄存器和内存单元(地址为(HL)、(IX + IND)、(IY + IND))

的任一位进行测试(是 0 还是 1),置位(使其为 1),复位(使其为 0)。位测试的结果反映在 Z 标志上,该位为 0 则 Z = 1, 为 1 则Z = 0 。置位、复位指令对所有标志均无影响。

8. 移位和循环移位指令。

助记符　Z80 的这一类指令多达 76 条,它们的助记符由 2～4 个字母按一定规律组成:

第一个字母:S——移位;R——循环移位。

第二个字母:L——左移;R——右移。

第三个字母:在移位指令中,L——逻辑移位(仅限右移);A——算术移位。

在循环指令中,C——小循环;无(或非 C)——大循环。

循环指令最后一个(第三或第四)字母为A时,表示是对累加器操作。

移位和循环指令能使一个字节内各二进制位的内容有规律地移动,以实现一定的处理目的。它们的操作内容可归纳如下:

所有向左循环或移位的指令,都是使每一位的内容向高端移动一位,最高位 D7 移入 C 标志位,即 C_y←原 D7,D7←原 D6……D2←原 D1,D1←原 D0。所有向右循环或移位的指令都是逐位向低端移动一位,最低位 D0 移入 C 标志位。

向左移动后的 D7,向右移动后的 D0,移进什么内容,是各种循环、移位指令的区别之所在。

(1)"大循环":将原来的C标志移入。如左移时,C_y←原D7……D0←原C_y。连进位标志构成九位的循环。

(2)"小循环":左移时 D0←原 D7,右移时 D7←原 D0;移入 C 标志位的内容不变。所以这只在八位内循环(C 标志只记存而不参加)。

(3)逻辑右移是 D7←0;算术左移是 D0← 0 。

(4)算术右移是原 D7 内容移入 D6,D7 仍保持不变。

除累加器专用循环指令不影响 Z、S 和 P/V 标志,其余循环、移位指令影响各标志位。

循环、移位指令的用途较多,例如:

第一,用于乘除法运算。从二进制数的原则可知,一个数各位左移等于将原数 × 2 ,右移则等于将原数 ÷ 2 。乘以某偶数,可用多次移位(× 2),乘以奇数可在移位后再加一个被乘数来完成。每次进行一个字节,进位和余数的问题可用不同的循环、移位指令来处理。这样,CPU"不会"的乘除法便可以用程序实现了。

第二,使一个数产生有序变化。例如 LASER　310 的键盘扫描,需要使六条地址线依次有一条为低电平,就是用循环指令使一个零位在地址值的低六位中移动,这样循环变化着的地址值即可用以实现扫描。

第三,将字节中的各位依次取出(通过 C 标志)或由一些二进制位组合成为字节。例如 LASER　310 磁带信息的读写都是以二进制位为单位进行的,必须以此来实现字节的分解与合成。

9. 通用算术指令。

CPL　累加器内容取反(各位变得同原来相反(0 ←→ 1)),对可检测标志均无影响。

NEG　累加器内容求补(取反后再加 1),影响各标志。

CCF　进位标志 Cy 求反(0 ←→ 1),仅影响 Z 标志。

SCF　进位标志 Cy 置 1，仅影响 C 标志。

10. 输入、输出指令。

助记符　IN(输入)；OUT(输出)

输入指令用来从指定的 I/O 口读取信息，输出指令用来向指定的 I/O 口写入信息。对于 LASER　310，只在同打印机、软盘驱动器等交换信息时使用。

I/O 指令中，IN　A，(N)对各标志均无影响，IN　r，(C)　影响 Cy 以外各标志，其余会影响 Z 标志。

11. 数据块传送、检索和 I/O 指令。

Z80 有丰富的数据块操作指令，给编程带来很大方便。

(1)数据块传送指令。

助记符　LDI(单步增址传送)；LDIR(整块增址传送)；LDD(单步减址传送)；LDDR(整块减址传送)

这些指令可以把<64K 字节的数据块传送到新的地址，如互不覆盖，源数据块内容不变。它们的符号指令不带操作数，实际上 BC，DE，HL 全部参加操作：HL 作为源地址指针，DE 作为目标地址指针，BC 作为传送记数器。使用数据块传送指令之前，必须给它们赋初值。

例如，要将从 8000H 开始的 160 个字节送往 A000H 开始的一片内存空间，则要用四条指令

```
LD   HL,8000 ;设定源起始地址。
LD   DE,A000;设定目标起始地址。
LD   BC,00A0;设定传送字节数。
LDIR         ;整块增址传送。
```

传送的具体过程是：第一步，(DE)←(HL)；第二步，HL＋1，DE＋1，BC－1；第三步，如 BC＝0，执行下一条指令，否则转回第一步。本指令执行结束时，HL 和 DE 分别指向源和目标数据块终址之后的地址。

如果传送过程中需作判断和处理，则用 LDI 代替 LDIR。它执行上述第三步时，不会回到第一步，即进入下一条指令。若需继续传送，用转移类指令转回 LDI 指令即可再传送一字节。不过其间各寄存器的指针值不能随意改变。

LDD、LDIR和LDI、LDIR 的不同之处仅在第二步，它们是 HL－1，DE－1，BC－1。显然传送的顺序是相反的。因此，HL 和 DE 的初值都应是数据块的终址(高端地址)。传送完毕后，分别指向数据块低端前一字节的地址。

若源数据块同目标数据块所占空间有一部分重叠，向内存高端传送时，应取减址方式；向内存低端传送时，应取增址方式。这样，传送到重叠区域时，其中的源数据已经送走，可以覆盖了。

这类指令对 C、Z 和 S 标志都无影响。

(2)数据块检索指令。

助记符　CPI(增址单步检索)；CPIR(增址整块检索)；CPD(减址单步检索)；CPDR(减址整块检索)

这类指令能在一个多达 64KB 的数据块内搜索指定的单字节代码。它们的单步、循环、增

址、减址等方式与数据块传送相似。由 AF、BC、HL 三对寄存器参加操作。A 中存检索目标代码,HL 作地址指针,BC 作计数器。执行前要给它们赋初值。如从 7AE9H 开始的 64 个字节内寻找 00H(语句结尾符),从低地址开始(增址)检索,程序如下:

```
LD  A,00
LD  HL,7AE9
LD  BC,0040
CPIR
```

其执行过程是:第一步,CP(HL),HL←HL+1,BC←BC-1;第二步,如 A=(HL)(Z)或 BC=0 则结束检索,执行下一条指令,否则转回第一步。

数据块检索指令执行后,从 Z 标志的状态便可知是找到目标代码(Z)还是未找到(NZ)。找到时,该代码的地址为 HL-1,未找到时 HL 则指向数据块之后一字节地址,A 的内容均不改变。检索指令只能查找一个字节代码,查找代码串须编程解决。

(3)数据块 I/O 指令。

助记符 INI(增址单步输入);INIR(增址整块输入);IND(减址单步输入);INDR(减址整块输入)。OUTI、OTIR、OUTD、OTDR(输出,分类同输入指令)

数据块 I/O 指令用HL指定内存地址,用 C 指定 I/O 口,用 B 为计数器。数据块最大只能有 256 字节。执行过程与数据块传送相类似。

12. 中断和 CPU 控制指令。

EI　　开中断,使 CPU 能响应 $\overline{INT}$ 中断请求。

DI　　关中断,使 CPU 不响应 $\overline{INT}$ 请求,但对"不可屏蔽中断"信号 $\overline{NMI}$ 无效。

RETI　　由中断返回,它除具有 RET 的功能外,还能使 CPU 响应下一级的中断。

RETN　　由不可屏蔽中断返回。

IM 0　　置中断方式 0。

IM 1　　置中断方式 1。即以 0038H 为中断服务子程序入口地址,LASER 310主要使用此方式。

IM 2　　置中断方式 2。

NOP　　空操作。可用来在程序中占位或延时(四个T周期)。

HALT　　CPU 操作暂停,直至响应中断或出现 $\overline{RESET}$ 信号为止。在 LASER 310 基本系统中,此指令一般不用。因为如未关中断,视频系统每 1/50 秒发来一个中断请求信号,便会结束暂停状态,如已关中断,又无 $\overline{RESET}$ 和 $\overline{NMI}$ 信号来终止它的作用,CPU 失去工作能力,只好关机重开。

13. BCD 码操作指令。

BCD 码是用 4 位二进制数码表示一个十进制数码的代码形式(如 32 用 0011 0010_{BCD} 表示,等等)。Z80 有一套对它进行处理的指令。它们是:DAA(BCD 码调整);RLD(BCD 码循环左移);RRD(BCD 码循环右移)。

第六章　软系统剖析

本章开始讨论标本机的程序系统——即通常所称的“软件”。软件是指挥计算机工作的信息的逻辑组织。计算机可以说是硬件和软件的统一体。一台只有硬件而没有软件的计算机是不能进行任何工作的。

软件的主体部分是机器指令的有序集合——程序，同时还包括执行程序所需要的数据。计算机软件可分为不同的层次。最基础的软件，应给使用者提供操纵机器的手段。这就是“操作系统(OS)”。操作系统中，包括控制 I/O 设备输入输出信息、程序的存贮配置和启动执行的“监控程序”，将用户以程序设计语言编写的源程序变成机器语言程序的“编译程序”或“解释程序”，用于装入、调试、编辑用户程序或其他通用性处理的“支持程序”。有了操作系统，用户就可以用规定的命令使计算机工作，而且可以在它的支持下输入、调试、运行、存贮自己的应用软件。

无论系统软件和应用软件，都必须放入内存贮器才能使用。在程序的运行过程中，还会产生一些工作数据或运算结果，也要存放在内存中。所以，计算机的内存空间，就是软件的工作场所和活动舞台。为了充分利用有限的内存空间，系统设计时便要对它进行划分，给各种程序和数据分配一定的地址范围。所以，内存的分配图也就可以作为软系统的结构图。

Z80 CPU 的直接寻址范围为 64KB。所以，LASER　310 系列机的最大内存范围也是 64KB。不同的机型和扩展配置，内存规模从 32K(200 型基本系统)到 64K(310 型加“64K卡”)。超过寻址范围的存贮器(例如“64K卡”的一部分存贮单元)，可以在软开关的控制下，与 64K以内的部分随时切换，作为性质介于内、外存之间的一种存贮器来使用。

LASER　310 系列微机各种配置虽然内存大小不同，但内存分配的原则是一样的。从低端到高端，可划分为以下部分：

0000H～3FFFH	V2.0 系统程序
4000H～67FFH	外接软件(如 DOS)
6800H～6FFFH	基本 I/O 接口
7000H～77FFH	显示文本区
7800H～7AE8H	V2.0 系统工作区
7AE9H～	用户 BASIC 程序区
	变量区
	空闲区
	堆栈
	字符串区
～内存终端	DOS 工作区(可无)

以上各区段的划分是通过两种不同的方式来确定的。0000H～77FFH内的各区段，由硬件决定，在第4章已经说明，而且其规模和配置，各型机均相同。7800H～内存终端这段空间，处在主存贮器中，总规模2K～34K，因不同型号和扩展卡而异，由系统软件分配使用。除7800H～7AE8H的系统工作区的规模和地址是固定的以外，其余部分，系统在运行中根据需要安排使用，用户也可以进行干预。

一、系统程序

（一）“BASIC V2.0”

LASER 310内存最低端的16KB空间，由一片ROM占据，其中固化着一个常驻内存的程序，版型为V2.0。它虽被称为“BASIC解释程序”，但其中也包含了控制信息输入输出的“监控程序”和帮助BASIC工作的“支持程序”。所以确切地说，它是一个“操作系统”。

V2.0系统程序，是在美国Microsoft公司的Level II的基础上，经修改而成。V2.0比Level II扩大了4KB程序文本。除监控程序完全不同外，在BASIC方面增加了一些功能，改变了一些原有的功能，也废弃了原来的若干功能。这些改动，使得程序文本的各个模块互相穿插，结构显得比较乱。按照地址的顺序，可以大致把V2.0的内容分为以下几部分：

0000H～0707H　系统初始化程序，监控程序的部分子程序。

0708H～1A17H　数学、支持程序和数据表。

1A18H～2EDBH　BASIC解释程序。

2EDCH～34A8H　监控程序。

34A9H～3FFFH　解释程序和监控程序的部分子程序。

以上几部分之间也并无截然的分野，各自又包含着很多相对独立和互相牵连的子模块。

由于系统程序常驻内存并将入口置于0000H，给用户使用BASIC语言带来很大的便利，不但免去每次从外设调入系统程序的麻烦，而且一开机CPU复位，便从0000H自动进入了V2.0系统，作好了接受用户BASIC程序或命令的准备。

V2.0是一个封闭式的系统，开机进入以后即在其中循环往复地运行，除了关机，是不会“停止”下来的。系统的最大特点是BASIC解释程序和监控程序以分时方式工作。

由于硬系统的结构，带来了两个问题。一是向VRAM写入显示信息，最好在视频消隐期进行，以避免在屏幕上造成杂乱光斑；二是非编码式键盘不能主动产生输入信息，必须以程序扫描检测。如果BASIC解释程序和监控程序轮流工作，比如执行完一个BASIC语句再执行监控程序一次，在BASIC工作期间的场消隐期就不能利用，而语句执行完毕积累的显示信息一个场消隐期又输出不了，还得等待下一个或多个场消隐期，两方面都造成时间浪费，同时，键盘的输入也必然会很不及时。最理想的办法是二者“同时”运行，但对于串行工作的CPU是不可能的。所以采取了分时切换的方法，以视频信号发生器的垂直同步信号$\overline{FS}$作为切换信号。系统的大部分工作时间是在执行BASIC解释程序。当显示下边框开始，$\overline{FS}$有效，CPU响应后，BASIC的工作被强行中断（不管其是否告一段落），监控程序投入运行，扫描键盘，将已准备好待输出的信息送往VRAM等。监控程序一次运行结束，便从中断返回，由刚才的断点继续执行BASIC解释程序，直至下一次中断信号的到来。系统的两部分之间，通过

系统工作区交换信息，以协同配合工作。在 PAL 制式下，上述分时切换过程每秒发生 50 次。这就使得每一个场消隐期都可被利用来显示，而且每 20ms 扫描一次键盘的密度，也不会放过任何一次按键动作。对于 BASIC，虽然一再被“打断”，但由于 Z80A 的速度很高，对于人的感官来说，似乎它的执行仍然是“一气呵成”的。

系统主循环的框架是 BASIC 循环，监控程序可看作是“镶嵌”在其中的，相对独立的单步周期。在 BASIC 的不同阶段，监控程序的工作内容也有所选择。二者的流程对照见表 6-1。

表 6-1

BASIC	监控程序
① 系统初始化。每次开机便自动进入，将系统工作区及各输出口初始化，以后一般不再调用。	1′ 因中断关闭，监控程序不工作。
② BASIC 输入程序。等待和接收监控程序的输入行(等待期间BASIC运行停滞)，并将其代号化。若是一行程序则送入用户程序区，转回②；若是直接命令则进入③。	2′ 输出显示，闪动光标，扫描键盘并将字符送入显示文本区。在BASIC执行②的等待期间，每次中断均反复执行2′。
③ 执行驱动程序。扫描 BASIC 程序或命令，根据语句命令的内容转入相应的工作子程序，工作子程序执行完毕返回③。如扫描到程序结束标志则返回②。执行中发现错误则进入④。	3′ 中断开放时，输出显示，扫描键盘，但不输入，不闪光标(INPUT除外)。如遇某些子程序关闭中断，则同1′。在 BASIC 执行③的过程中，每次中断反复执行3′。
④ 错误处理程序。报告出错信息以后转回②。	4′ 输出显示，扫描键盘但不输入，不闪光标。

(二) 磁盘操作系统——DOS V1.2

从经济的角度考虑，LASER 310不以价格较高的磁盘驱动器作为基本外设，V2.0 也不具备磁盘操作功能。但为了便于系统的扩展，为 V2.0 以外的系统软件保留了 10KB 的地址空间，可通过扩展插口加插软件扩展卡来使用。地址为 4000H～67FFH。LASER 310 的磁盘操作系统——DOS V1.2，就是以“磁盘控制卡”的形式提供的。

DOS V1.2 固化在 ROM 中，主要内容是磁盘操作命令的解释程序和磁盘驱动器的驱动程序。V2.0 与 DOS V1.2 之间为软件切换方式。在 V2.0 系统初始化的后期，将检测是否接有 DOS 卡(详见下一章)。如接有 DOS 则进入执行，对 DOS 系统初始化，并设置在对

BASIC 解释执行中转入 DOS 的软接口，以便执行 DOS 命令。本书对 DOS V1.2 不作详细讨论。

二、V2.0 系统工作区

主存贮器最低端的 7800H～7AE8H，由 V2.0 系统程序辟为自己存放各种工作数据的场所。称为"系统工作区"。它的用途非常广泛，是系统程序工作的重要支柱。

V2.0 的各个子程序之间，系统与用户程序之间，需要经常互通信息，方能协调一致地工作。例如，变量区是紧接在用户 BASIC 程序的末尾的，随着程序增删，它的地址也不断地浮动，给存放和寻找变量带来困难。所以每一次增删程序后，系统都将新的变量区起始地址和结束地址记录在系统工作区的两对固定的存贮单元中。这样，任何时候别的子程序都能方便地找到变量区了。系统工作所必需的各种指针、参数、标志和入口向量等，各有固定的存放地址。它们的作用很像"信箱"或"留言簿"，所以又称作"通讯区"。

系统工作区的一部分空间用作输入输出、数据处理的缓冲区和暂存区，相当于"备料间"或组装、拆卸的"工作台"的作用。

系统工作区由系统程序使用并加以保护。用户使用 BASIC 语言时，可以不予理会。用 POKE 语句能够直接改变其中内容，若改变不当，会使系统工作混乱，甚至造成不可挽回的后果。但在对系统工作区有所了解的基础上，有目的地改变某些信息，就能左右系统的工作。不了解系统工作单元的用途，很难读懂系统程序，而一些单元的作用，又只有在对系统程序有关部分进行剖析后才能真正理解。

（一） 软接口——转移向量和地址

这是一些可"填写"的地址表，主要用作固化了的系统程序和以后的扩展系统或用户程序之间的软接口。它们像一些"联络站"。固化系统工作时将访问它们。只要把需要提供主系统访问的地址填入其中，即可实现中转。这就比固定地址方式有更好的可扩充性。

1. 零页调用子程序入口向量：V2.0 零页的 RST 指令入口处，各固化着一条指向系统工作区的指令，如在 RST 10H 指令的入口 0010H，是指令 JP 7803H。系统初始化时便将几个常用的系统功能子程序入口向量存放在这里，使得它们可用简短快速的 RST 指令调用。

7800/7802H　RST 08H 向量(JP 1C96H "取下一字符"子程序)

7803/7805H　RST 10H 向量(JP 1D78H 字符检测子程序)

7806/7808H　RST 18H 向量(JP 1C90H 比较 HL 和 DE 寄存器子程序)

7809/780CH　RST 20H 向量(JP 25D9H 测试数据类型子程序)

780C～7811H　为 RST 28H，RST 30H 保留的单元，V2.0 未使用。RST 38H (中断服务程序入口)的向量已固化在 ROM 的 0038H，不需中转。

2. 键盘检测程序入口地址(7816/7817H)。

系统初始化已置为 2EF4H。

3. 中断服务程序出口向量(787DH～787FH)。

监控程序运行时首先访问这个地址，若设置为指向用户中断服务子程序入口的向量，即可在系统中断时优先得到执行。

4. 用户机器语言子程序(USR 函数调用)入口地址(788E/788FH)。

用户置入子程序入口地址后,用 X=USR(N)语句即可调用此程序。

5. "磁盘 BASIC"出口向量表(7952H～79A5H)。

这是 Level II 系统用来执行扩展的"磁盘 BASIC"命令的,每三字节存放一个"磁盘 BASIC"子程序入口向量。V2.0系统未用,初始化时全部置成"JP 012DH"。如用户使用了有关代号,便经这里转去显示"DISK COMMAND? SYNTAX ERROR"错误信息。

6. "DOS 出口"向量表(79A6H～79E4H)。

这是 Level II 系统用来同 TRSDOS 沟通的软接口。某些 BASIC 子程序要访问这些地址,当接口被 DOS 预置后,即由此转入 DOS 的有关部分执行。LASER 310 的DOS V1.2 没有使用这种方式。V2.0 初始化时已把它们首字节全部置为 C9H(RET),使对这些地址的访问立即返回。

(二) 专用子程序

这是系统工作区比较特殊的用途。这些子程序中留有空字节,在"填入"参数后加以调用,所以不能放在 ROM 中。

7880H～788DH　　除法运算用的减法子程序。

7893H～7895H　　(INP 语句调用的子程序(IN A,□H　　RET)。

7896H～7898H　　(OUT 语句调用的子程序(OUT □H,A　　RET)。

(三) 暂存、缓冲区

1. 字符串库(78B5/78D2H)。

用以存放字符串运算的中间结果和准备输出字符串。

2. 工作寄存器(791DH～792EH)。

进行单精度或双精度运算时,CPU 的寄存器不够使用,就以工作寄存器(WRA)存放操作数、中间和最终结果。其中 791DH～7924H 为 WRA1,7927H～792EH 为 WRA2。单精度运算用 WRA1 存放一个操作数,另一个存放在寄存器 BCDE 中,结果存于 WRA1。双精度运算的两个操作数分别存放在 WRA1 和 WRA2,结果存于 WRA1 中。

3. 打印缓冲区(7930H～7949H)。

存放待打印输出的行,供打印机驱动程序取用。

4. 输入缓冲区(79E8H～7A9CH)。

用来存放 BASIC 输入程序从显示文本区取回的一行字符,并用作输入语句代号化的工作场所,从 79E6H 开始,暂存按 BASIC 格式整理好的程序行,以便送入用户程序区或直接解释执行。INPUT 子程序、LIST 子程序和全屏幕编辑时也用输入缓冲区作为工作场所。V2.0 接受输入行时只使用 64 个字节,所以每行只能敲入 64 个字符。但如用户人为地使程序行字符数超出 64 个,LIST 将代号化语句还原时,也使用其余的空间。

5. 磁带文件名暂存区(7A9DH～7AACH)。

存放用户给出的磁带文件名,以便记录或核对。

6. 显示缓冲区(7AB2H～7AD1H)。

BASIC 产生的显示代码先存放在这里,监控程序由此处取出送往显示 RAM。

(四) 系统指针、标志和变量

7818/7819H	显示状态控制单元。7818H为0时绿底黑字，非0时反白显示；7819H保存7818H内容，二者不同时，监控程序将改变原来的显示状态。
781E/781FH	装入的磁带文件起始地址
7820/7821H	与光标当前屏幕位置对应的显示区地址
7823／7824H	磁带读／写字节计数器
7825H～7829H	打印机控制块，由打印子程序取用。其中
7825H	打印机设备类型码（06H）
7826/7827H	打印驱动程序地址（058DH）
7828H	每页行数（43H）
7829H	打印行记数器
7836H	键盘检测时存第一次扫描到的键码
7837H	键盘检测时存第二次扫描到的键码
7838H	键盘功能状态字，各位的作用见“监控程序”一节。
7839H	I／0控制字。D0=0 等待键盘输入，D0=1 键盘输入行结束；D1=0 键码非0，D1=1 键码为0；D2=0 RETURN键结束输入行，D2=1 BREAK键结束输入行；D3=D6=0 CLOAD调用；D3=0,D6=1，CRUN调用；D3=1,D6=0 VERIFY调用；D4=0 BASIC输入程序调用，D4=1 INPUT子程序调用；D5=0 非接受输入子程序调用，D5=1 接受输入子程序调用。
783AH	按键延时计数器，计数达到规定次数后即作为按第二次键。
783BH	输出锁存器副本
783CH	暂存光标所在的字符
7841H	光标闪烁延时计数器，控制光标闪烁周期；初值为10H，每次响应中断计数减1。
7842H	按键所在的行数
7843H	按键所在的列数
7844/7845H	按键所在行的扫描地址
7846H	图符颜色代码（含于D6～D4中）
784CH	磁带输入提示信息（“WAITING”等）显示控制单元（0—显示提示信息，非0—不显示提示信息，可避免显示字符破坏高分辨率画面。）
7899H	暂停时打入的键码
789AH	当前错误的代码（02H～2CH），由RESUME子程序清为 0。
789BH	打印行的字符计数
789CH	选用的输出设备代码（01H=打印机，00H=显示器，FFH=磁带机）
789DH	显示器每行字符数（40H）

789EH	打印机每行字符数
78A0/78A1H	字符串区的低端地址（由CLEAR子程序划定），也就是堆栈的底。
78A2/78A3H	当前执行语句的行号值（直接命令为FFFFH）
78A4/78A5H	BASIC程序的首地址（系统安排为7AE9H）
78A6H	光标在行中的位置（00H～3FH）
78A7/78A8H	输入缓冲区首地址（79E8H）
78AA～78ACH	随机数的“种子”值。系统初始化和RANDOM子程序将 CPU刷新寄存器 R的当前值存入78ABH。
78AEH	寻找变量的目的标志（00H—寻找变量，01H—建立变量）
78AFH	WRA1中数据类型的标志（02H—整数，03H—字符串，04H—单精度数，08H—双精度数）
78B0H	表达式计算中存当前算符代号，代号化时存放进入DATA语句标志。
78B1/78B2H	内存最高地址，由系统初始化程序测定，用户可重新设定。
78B3/78B4H	字符串库指针（下一个可用单元地址）
78D3H	送入串区的字符串的长度（00H～FFH）
78D4/78D5H	送入串区的字符串的地址
78D6/78D7H	字符串区指针（下一个可用单元的地址）
78D8/78D9H	表达式计算中存放算符的地址
78DA/78DBH	已读过的最后一个DATA语句的行号
78DCH	FOR标志（执行FOR语句时为64H）
78DDH	输入阶段标志（00H—进入程序输入阶段）
78DEH	读数据标志（00H—READ有效，01H—INPUT有效）
78DFH	存放执行程序的扫描起始地址和循环变量地址
78E1H	AUTO标志（0—非AUTO，非0—AUTO）
78E2/78E3H	AUTO的当前行号
78E4/78E5H	AUTO的行号增量
78E6/78E7H	在程序输入阶段存放已代号化语句的地址，在程序执行阶段存放当前执行语句之前一个字节的地址。
78E8/78E9H	当前语句开始执行前的堆栈指针
78EA/78EBH	出错语句的行号
78EC/78EDH	同上
78EE/78EFH	出错语句前一字节的地址
78F0/78F1H	ONERR（逢错转移）的目标地址
78F2H	错误捕获标志（非0—捕获到错误，0—未捕获错误或捕获后已经

RESUME返回）

78F3H	打印缓冲区中小数点的位置；表达式计算中存下一个代号的地址。
78F5/78F6H	程序执行结束、中断或出错时的最后一个行号
78F7/78F8H	在程序执行结束、中断或出错前最后执行过的地址
78F9/78FAH	简单变量区的首址，即BASIC程序的终址（最后字节地址＋2）。
78FB/78FCH	下标变量区的首址
78FD/78FEH	空闲区首址，即变量区终址。
78FF/7900H	READ读数指针（指向下一次读数的开始地址）
7901H～791AH	变量类型表。共26个字节，每个对应于英文字母表的一个字母，存放该字母打头的变量类型代码（见78AFH），系统初始化为04H。
791BH	跟踪标志（0—不跟踪，非0—跟踪）
791CH	数学运算使用的暂存器
7925H	存放运算结果的符号
7926H	双精度加法使用的暂存器
794AH	双精度除法使用的暂存器
7AAEH	光标在行中的下一个位置
7AAFH	显示缓冲区计数器（送出显示完毕为0）
7AB0/7AB1H	显示缓冲区指针
7AD2H～7AD6H	图形COPY工作单元。并作以下用途：
7AD2H	磁带文件类型（F0H—T类，BASIC文本文件；F1H—B类，机器语言文本文件）。也用来存放SOUND语句的音高参数等。
7AD3H	暂存从磁带读读入的一个字节
7AD6H	磁带文件名（包括结尾符00H）的长度
7AD7H～7AE6	显示行性质标志，每个字节表示屏幕一行的属性。

三、BASIC 程序及变量

（一） BASIC 程序

BASIC 程序是读者都很熟悉的。无论是书写、键盘输入还是屏幕显示，它都由一个个字符组成。例如下面这个小程序

```
10  PRINT"OK":SOUND20,8
20  END
```

在内存里面，是不是就将这样 28 个 ASCII 码（包括空格）连续存放呢？当然，这样做未尝不可，但当程序较长、分支较多时，这种过于简单的数据结构将使得查找、执行和修改程序都很困难。一个操作系统，必须为 BASIC 程序选择一种简洁高效的数据结构。同样是 BASIC

语句，在各种系统里可能具有完全不同的存贮格式。

V2.0 系统将 BASIC 程序以“链表”结构存放。程序行按行号大小呈升序排列，每个程序行(可以包含多个语句)为一个“链节”。每行程序的结构是：

开头用两个字节作为“链指针”，放置后续行的首地址；这是一个 16 位二进制数，低 8 位在第一字节，高 8 位在第二字节。链指针是环环相扣的，第一个的内容是第二个的地址，第二个的内容是第三个的地址……沿着这条链便可跳过所有无关的语句体，迅速达到任何一个目标语句。

第三、四字节存放行号值，也是用 16 位二进制数表示。行号是语句的标识。系统在查找一个程序行时，沿着指针链逐步跳进。每到达一行后就取后面两字节的行号值比较，若小于目标行再跳向下一行……直到找到目标行或证实无此行号为止。

从第五字节开始是语句内容。其中不同的语法成分又采取不同的形式。语句、命令、函数的保留词和运算符，均为“代号”形式。例如“PRINT”这个词，在送入程序区以前先由系统将它“代号化”，变为单字节的代号 B2H 存放。这样不仅节省存贮空间，而且解释执行时凭代号就很容易计算出工作子程序的入口，不必再花费时间来识别“PRINT”这个词语，可以加快执行的速度。代号都是≥80H 的，与≤7FH 的 ASCII 字符码容易区别。不过，反白字符和图案符号的显示代码也≥80H，因而规定它们只允许在引号内的字符串中出现，以免同代号混淆。其余字母、符号和数字，一律以 ASCII 码存放。行内语句之间以冒号(：)间隔。

程序行最末一个语句体之后，以一个零字节(00H)作为行的结束标志。语句体中不可能含有 00H(数字 0 的ASCII 码是 40H)，所以遇到零字节就知道本行结束，后边紧接着是下一行的链指针。

整个程序语句表最末一个语句的结束符之后，以两个零字节表示程序结束。扫描发现链指针值为 0000H 时，就意味着程序结束了。

前面那个小程序在内存里的存贮格式如图 6-1 所示。

7AE9	7AEA	7AEB	7AEC	7AED	7AEE	7AEF	7AF0	7AF1	7AF2	7AF3	7AF4
FA	7A	0A	00	B2	22	4F	4B	22	3A	9E	32
(指针7AFA)		(行号10)		PRINT	“	0	K	”	:	SOUND	2

7AF5	7AF6	7AF7	7AF8	7AF9	7AFA	7AFB	7AFC	7AFD	7AFE	7AFF	7B00
30	2C	38	00	FF	7A	14	00	80	00	00	00
0	,	8	结尾符	(指针7AFF)		(行号20)		END	结尾符	程序结束标志	

图 6-1

存贮格式由系统产生，用户只需从键盘按字符敲入程序。输入行在缓冲区被加工成标准格式后，系统便把它移入程序区存放起来。存放程序的起始地址是由“BASIC程序区首址指针”(78A4/78A5H的内容)规定的。V2.0系统初始化时确定为7AE9H。用户如果改变了这个指针的值，实际上也就可以改变程序区的首地址。程序区的结束地址是会随着程序的增删而变化的。系统会随时将程序结束标志之后那个单元的地址记入程序终址(即变量区首址)指针(78F9/78FAH)。

程序行的存贮顺序，不按输入先后，而由系统按行号大小依次存放。从中间加入程序行时，更高行号的程序段将由系统将它向高端移动，以腾出存贮新行的空间。如新行行号与原有行号相同，则删去原行而代之以新行。只输入行号时，便将该行从程序中删除，以后的程序段前移填补所占空间。经上述变动后，虽然保留行的语句体不变，但因存贮地址移动，链指针的地址值也不符现实情况，需要从头修正链指针。这些工作均由系统自动进行，无需用户过问。

执行LIST命令将程序列表时，系统逐行将上述格式的语句取入缓冲区，进行代号化的逆过程，将它复原成输入时的字符串格式显示出来。执行NEW命令后，系统将对程序区进行初始化，把程序区开头两个字节清为零，即程序结束标志，并把第三个字节的地址存为程序终址指针。这样，对程序区的访问都被拒之门外，等于清除了程序区。其实，内存的其余内容并未改变。所以，采取一定手段，还可以把程序“救”回来。

对程序区的结构有所了解以后，用户在必要时就可以撇开系统自行(用POKE或其他手段)改变程序区的内容以达到某些特殊效果。但必须遵守上述规定，注意每一个字节的不同意义，加长或缩短程序行，移动程序的地址，都必须相应修改链指针，否则将会使程序面目全非。

(二) BASIC变量

BASIC程序中的数据，除一些直接数外，大量使用变量。每一个变量一经赋值，便需要存贮起来，以便引用。

变量区是在程序或命令执行时，用来存放变量的空间。它紧接在程序区之后，又可分为两个区域：简单变量区和下标变量区。它们的地址是随着程序的增删和变量的增减而浮动的。其地址的每一次变化都必须记入通讯区，简单变量首址指针在7AF9H/7AFAH，下标变量首址指针在7AFBH/7AFCH。变量区的结束地址即空闲区的首址，指针在78FDH/78FEH。变量区尚未使用时，上述三者内容相同。

变量按它首次被定义、赋值和引用的先后次序存放，一经存入，便只能改变它的值，而不能个别“清除”。CLAER和RUN命令，将把变量区三个指针都恢复到程序的终址，等于全部清除了原有变量。

为了便于搜索，变量表也采取链表数据结构。但它和程序链表不相同的是，以本变量占用的字节数作为链指针。因为每种类型的变量长度都是确定的，所以这也就是变量类型标志。搜索变量时通过加运算，便能沿着链结构跳进。

1. 简单变量的存贮格式。

第一个字节为类型标志，即给变量值所分配的字节数，可为2、3、4、8四个数之一。

第三、四字节为变量名，以ASCII码表示。变量名的第一个字母放在第三字节，第二个字

母放在第二字节，以后更多的字母或数字均不存入，所以，前两个字符相同的变量都被系统视作同一变量。仅用一个字母作变量名时，第二字节为零字节。

变量的值从第四字节开始。数值变量以二进制形式，按低位字节在前、高位字节在后的顺序存贮。整型变量值占两字节，高字节的最高位 D15 为符号位（0 ——正，1 ——负），D14～D0 存绝对值（0～32767，如符号位为 1，绝对值为 0，其值为 8000H = -32768）。双精度（V2.0 未用）和单精度型变量的最高字节存放"偏移指数"，即二进制尾数小数点须移动的位数，其最高位表示移动方向（0 ——右移，1 ——左移），其余七位表示移动位数（移动位数的补码）。尾数分别占三（单精度）和七（双精度）字节。尾数只是二进制值的有效数码，从最高位开始依次存放，由尾数和指数共同决定值的大小。尾数的最高位同时也是符号位。由于尾数的第一位有效数字总是 1，所以存放正数时将符号位变成 0，取用时再恢复成 1，符号位未变的，便是负数了。

字符串变量的"值"并非字符的代码，而是用来给它赋值的那个字符串所在的地址。在程序中用 LET 语句（如 A $ = "ABCDE"）赋值的串变量，用该字符串在程序语句内的地址；如由 INPUT 语句输入，或字符串函数、字符串运算结果赋值的串变量，则为该字符在字符串区内的存贮地址。

2. 下标变量的存贮格式。

下标变量的存贮格式比较复杂。前三个字节与简单变量相同。第四、五字节是链指针，存放与下一数组之间距离的单元数，以便寻找变量时跳过无关数组的大片空间，加快速度。第六字节是下标的维数。再以下存放每一维所允许的下标个数，由于 0 也可作下标，所以它们等于该维的定义值加 1，每维占两字节，按下标的右→左次序存放。最后，依次存放每个元素的值，排列顺序从全部下标为 0 的那个的元素开始，从左下标到右下标，逐个下标依次增值（增量 = 1），直到最右边一个下标达最大定义值的那个元素为止。其间不加任何标识。例如，数组 X(2,4) 各元素值的存贮顺序是：

X(0,0) X(1,0) X(2,0) X(0,1) X(1,1) X(2,1)……X(0,4) X(1,4) X(2,4)。

数组的存贮空间在执行 DIM 语句时便已全部开辟出来，初值均存入 0。所以定义以后不充分利用，会造成内存的浪费。

变量的存贮格式这样复杂，目的是为了便于系统使用它们。如果不用变量，而是直接在内存单元中存贮数据，显然可以节省不少空间，但查找起来就不方便了，因此只宜在数据比较简单时采用。

四、字符串区和堆栈

系统的以上各个部分是按内存低端→高端的方向排列的，以下三个部分则不同，按高端→低端方向安排。

V2.0 系统初始化时，要测试内存物理终端的地址。如系统接有 DOS，则将这个实测地址减 311 字节，作为内存终端指针，使高端这片空间受到保护，用作 DOS 的工作区，并将它的首址存入变址寄存器 IY，以便使用。如果未接 DOS，系统则从内存终址开始设置字符串区。

(一) 字符串区

字符串区用来存放一部分字符串。程序中带引号的字符串不使用串区。它只存贮在程序中"找不到"的那些字符串，即字符串变量的值。字符串区占用内存最高端(在 V2.0 基本系统为机器的"硬终端"，在有 DOS 时为系统设定的"软终端")的一片空间。串区的规模由系统或用户规定，并不随字符串的多少而浮动。系统初始化时，已把它辟为 50 字节。BASIC 的 CLEAR 语句，可用来规定字符串区的字节数，如 CLEAR 1000。显然，开辟串区时应该精打细算，因为它的多余空间并不能挪作别用。而当空间不够时，固然可以重新设定，但原有的内容将全部失效。

字符串从高端向低端存放，每个字符串以零字节为结尾符号。串区有两个指针，其最低端地址存在 78A0/78A1H，下一个可用单元的地址存在 78D6/78D7H。字符串由系统存入，每次均对剩余空间进行测试，空间不够则报告"OUT OF STRING SPACE"。

（二） 堆栈

堆栈同时作两方面用途，既供系统程序本身的 Z80 机器指令使用，又由系统用来存放 BASIC 的 FOR 参数和 GOSUB 参数。系统初始化过程中，将堆栈建立在字符串区之上。串区低端地址 - 2 作为栈底，既存入 Z80 的堆栈指示器 SP，又存入 V2.0 的堆栈栈顶指针78E8/78E9H。每次执行 NEW、CLEAR 和 RUN 都使栈指针复位到此。

V2.0 在执行 FOR 语句时，要把 18 个字节的一组参数压入堆栈，先后次序(存贮地址由高向低)是：FOR 代号(81H)，循环体第一个语句的行号（2 字节)，FOR 语句的行号(2 字节)，循环终值 (TO 参数，4 字节)，增量 (STEP 参数，4 字节)，数据类型(01H——单精度，FFH——整型)，增量的符号(01H——正，FFH——负)，循环变量的存放地址（2 字节)。

每次执行 NEXT 语句，都要到堆栈中寻找循环变量地址相同的数据块，取出有关数据，控制增量和转移。当循环结束时，系统将把堆栈指针下移 18 字节，将这块数据从堆栈中排除。所以，执行 BASIC 程序时如不经过 FOR 语句，或在循环结束以后，直接转入循环体内，遇到 NEXT 时都会产生"NEXT WITHOUT FOR"(即找不到 FOR 数据块)的错误。

执行 GOSUB 语句，也会往堆栈中压入一组七个字节的数据块，依次为：GOSUB 语句的地址(2 字节)，GOSUB 语句的行号(2 字节)，GOSUB 代号(91H)，执行驱动程序的地址(1D1EH)。实际上，在 GOSUB 语句执行完毕，最顶上的两个字节已经弹出，以转入执行驱动程序。堆栈中仅保存 5 个字节。

（三） 空闲区

在变量区的终址和堆栈栈顶之间，是系统尚未使用的空闲区。在程序输入和运行时，一般是从两头向空闲区"侵入"。为了维持程序的"活力"，一定的空闲区是必须的。如果空闲区已经没有足够使用的空间，系统发现后便会报告"OUT OF MEMORY"(内存溢出)，而不会将数据越界存放。

五、系统的初始化

计算机在没有开机的时候，是一台纯粹的硬设备。打开电源以后，给 ROM 中固化的系统程序注入了生命力，可以将存贮的机器指令送给 CPU 执行了。但这时软系统的其余部分还并不存在。例如，内存尚未得到划分，各种系统变量和指针未赋初值，各缓冲区内是一些不需要的信息等等，使得系统仍无法正常工作。所以，一个系统要投入运行，首先必须把它的基本

结构建立起来，并使其进入可正常工作的初始状态。这就称为“系统初始化”。可用一个简单的例子说明系统初始化的必要性。开机的时候，显示 RAM 各存贮单元的状态是任意的，显示发生器便会将它们作为显示代码，使屏幕上显示出杂乱的各种字符和图符，不仅难看，而且给用户的操作增加了麻烦。如果在初始化中将显示 RAM 每个字节都装入空格码，便会有一块“干干净净”的屏幕供你使用了。细心的用户想必已经留意到，LASER 310 开机的瞬间的确有一闪而过的“抹光”屏幕的动作，那就是初始化的痕迹。

不仅开机时需要初始化，由一个软系统转换为另一个不兼容的软系统，同样要进行系统初始化。既然初始化时本系统尚未完全建立，显然只能利用现有系统的手段。对于 V2.0，就是简单地使用 Z80 机器指令、硬件地址和本身的子程序进行。

机器打开电源的一瞬间，复位电路使 CPU 复位，程序计数器 PC 被清零。CPU在第一个取指令周期从 0000H 单元开始取出指令。所以把 0000H 作为系统初始化程序的入口，每次开机先使 V2.0 系统初始化。在机器工作过程中，转入或调用 0000H 入口，都会使系统回到初始状态。用户机器语言程序运行时，偶尔会莫明其妙地使系统初始化，就是因编程有错导致误入了 0000H 的缘故。

V2.0 初始化的任务是：第一，使基本外设和 I/O 接口进入初始状态；第二，测试内存大小，划分内存；第三，给系统工作区各单元置入各自应有的初始值；第四，检测有无外接的软件，如有则转入执行；第五，清除屏幕，显示系统标志，准备接受输入。

系统初始化程序分为地址不连续的四个程序段，下面是它们的反汇编文本及其注释。

```
0000-  F3       DI             ；关中断，暂不执行监控程序。
0001-  AF       XOR A          ；累加器清零。
0002-  320068   LD (6800),A    ；将00000000B送到I／O地址，使显示为低分辨
                                 率、绿色背景的文本模式。
0005-  C37406   JP 0674        ；因以下是零页调用入口，转移。

0674-  00       NOP            ；此二单元未用。
0675-  00       NOP
0676-  21D206   LD HL,06D2     ；固化在ROM中的系统参数源数据块首址。
0679-  110078   LD DE,7800     ；需送入的目标（系统工作区）首址）。
067C-  013600   LD BC,0036     ；本数据块字节数。
067F-  EDB0     LDIR           ；将初始数据送入系统工作区(此后DE=7836H)
0681-  3D       DEC A          ；A作为循环计数器。
0682-  3D       DEC A          ；每次减2，直至A=00H。
0683-  20F1     JR NZ,0676     ；将传送过程重复128次。
0685-  0627     LD B,27        ；设定循环次数为39次。
0687-  12       LD (DE),A      ；依次将系统工作区7836H～7862H单元清零。
```

```
0688-   13        INC DE          ；目标地址加1。
0689-   10FC      DJNZ 0687       ；未做完39次则循环。
068B-   C37500    JP 0075         ；转入下一个程序段。
0075-   108078    LD DE,7880      ；系统工作区内又一目标首址。
0078-   21F718    LD HL,18F7      ；固化的源数据块地址。
007B-   012700    LD BC,0027      ；传送字节数。
007E-   EDB0      LDIR            ；把又一批初始数据送入系统工作区。
0080-   21E579    LD HL,79E5      ；指向输入缓冲区往前第三字节。
0083-   363A      LD (HL),3A      ；在第一字节中放入冒号代码。
0085-   23        INC HL          ；指向下一单元。
0086-   70        LD (HL),B       ；将第二字节清零。
0087-   23        INC HL          ；指向下一字节。
008B-   362C      LD (HL),2C      ；将逗号代码放入第三字节。
008A-   23        INC HL          ；令HL＝输入缓冲区首址（79E8H）。
008B-   22A778    LD (78A7),HL    ；置为输入缓冲区指针。
008E-   112D01    LD DE,012D      ；令DE＝012DH（显示“磁盘命令错误”信息的
                                    子程序地址）。
0091-   061C      LD B,1C         ；为预置磁盘BASIC出口设定循环计数器。
0093-   215279    LD HL,7952      ；指向“磁盘BASIC出口地址”的首字节。
0096-   36C3      LD (HL),C3      ；将JP指令操作码（C3H）放入首字节。
0098-   23        INC HL          ；指向下一字节。
0099-   73        LD (HL),E       ；将2DH放入第二字节。
009A-   23        INC HL          ；指向下一字节。
009B-   72        LD (HL),D       ；将01H放入第三个字节。以上三个字节组成指
                                    令“JP 012DH”。
009C-   23        INC HL          ；指向下一个入口的第一个字节。
009D-   10F7      DJNZ 0096       ；重复进行，置完28个入口为止。
009F-   0615      LD B,15         ；为预置DOS出口设定循环计数器。
00A1-   36C9      LD (HL),C9      ；将RET指令(C9H)放入首字节以关闭此出口。
00A3-   23        INC HL          ；跳过应为DOS内部地址的两字节。
00A4-   23        INC HL
00A5-   23        INC HL          ；指向下一个DOS出口的首字节。
00A6-   10F9      DJNZ 00A1       ；循环，至31个出口完全关闭为止。
00A8-   21E87A    LD HL,7AE8      ；指向用户BASIC程序区前一单元。
```

```
00AB-  70       LD (HL),B      ；存入00H，以便执行驱动程序识别。
00AC-  31F879   LD SP,79F8     ；临时设置堆栈，因下面CALL指令需使用。
00AF-  CD8F1B   CALL 1B8F      ；利用BASIC指针初始化程序，设定堆栈、字符
                                 串库等指针，以备初始化阶段临时使用。
00B2-  CDC901   CALL 01C9      ；调用清屏子程序清除屏幕。
00B5-  00       NOP            ；以下九个单元未用。
  ⋮
00BE-  1804     JR 00C4        ；跳过4字节。
  ⋮                            ；以下4字节不用。
00C4-  214C7B   LD HL,7B4C     ；以下测试内存容量。先给出最低地址。
00C7-  23       INC HL         ；指向下一地址。
00C8-  7C       LD A,H         ；地址高8位放入A。
00C9-  B5       OR L           ；与低8位相“或”以测试下一地址是否为0。
00CA-  281B     JR Z,00E7      ；如下个地址为0，则说明已测试到最高可能地
                                 址FFFFH，转去结束测试。
00CC-  7E       LD A,(HL)      ；取待测单元的内容。
00CD-  47       LD B,A         ；保存在B中。
00CE-  2F       CPL            ；取待测单元内容的反码为测试码。
00CF-  77       LD (HL),A      ；将测试码存回该单元。
00D0-  BE       CP (HL)        ；检验是否有效地写入该单元了。
00D1-  70       LD (HL),B      ；及时归还该单元原来的内容。
00D2-  28F3     JR Z,00C7      ；如写入有效（两者相等），则证明它是RAM，
                                 转入测试下个单元。
00D4-  1811     JR 00E7        ；如写不进去则非RAM，转去结束测试。
  ⋮                            ；以下 17字节不用。
00E7-  2B       DEC HL         ；退到已测有效存贮器的最后单元地址。
00E8-  11147C   LD DE,7C14     ；令DE＝必须具备的最小存贮器地址。
00EB-  DF       RST 18         ；与实有地址值（HL中）比较。
00EC-  DA7A19   JP C,197A      ；如HL<DE则转去显示“内存溢出”错误信息。
00EF-  11CEFF   LD DE,FFCE     ；令DE＝-50,以便设定字符串区初始容量。
00F2-  22B178   LD (79B1),HL   ；将测得的最高地址置为内存终端指针。
00F5-  19       ADD HL,DE      ；从内存终端后退50字节。
00F6-  22A078   LD (78A0),HL   ；置为字符串区首址指针。
```

```
00F9-  CD4D1B   CALL 1B4D      ；正式使BASIC各项指针初始化。
00FC-  CD8434   CALL 3484      ；将监控程序部分指针初始化。
00FF-  210F01   LD HL,010F     ；HL＝“VIDEO……V2.0”字符串地址。
0102-  CDA728   CALL 28A7      ；调用显示子程序将其显示在屏幕上端。
0105-  ED56     IM 1           ；置中断方式 1，以0038H为中断服务程序入
                                 口，以便利用中断执行监控程序。
0107-  C38E06   JP 068E        ；转入下一个程序段。

068E-  210040   LD HL,4000     ；指向外接软件第一区段入口地址。
0691-  CDA406   CALL 06A4      ；检测开始的四个单元是否 AAH,55H,E7H,18H
                                 （外接软件标志，两两互为反码）。如有此
                                 标志则从第五字节开始执行此软件。
0694-  210060   LD HL,6000     ；指向外接软件第二区段入口地址。
0697-  CDA406   CALL 06A4      ；检测（同上）。
069A-  210080   LD HL,8000     ；指向外接软件第三入口地址。
069D-  CDA406   CALL 06A4      ；检测（同上）。
06A0-  FB       EI             ；开中断，使监控程序能开始工作。
06A1-  C3191A   JP 1A19        ；无外接软件就转入BASIC输入程序。

3484-  CDA03F   CALL 3FA0      ；测试此时是否按着CTRL键，若是则变为反白
                                 显示模式（见下）。
3487-  3E20     LD A,20        ；基本I／O口输出初值(D5=1,D0=0,以便发声)。
3489-  323B78   LD (783B),A    ；存为输出副本备用。
348C-  320068   LD (6800),A    ；输出。
348F-  3E3C     LD A,3C        ；按键延时计数器初值。
3491-  323A78   LD (783A),A    ；存入计数器。
3494-  3E10     LD A,10        ；光标闪烁延时计数器初值。
3496-  324178   LD (7841),A    ；存入计数器。
3499-  AF       XOR A          ；A清零。
349A-  32AF7A   LD (7AAF),A    ；显示缓冲区计数器清零。
349D-  21B27A   LD HL,7AB2     ；HL=显示缓冲区首址。
34A0-  22B07A   LD (7AB0),HL   ；置为显示缓冲区指针初值。
34A3-  3EC9     LD A,C9        ；A=RET指令机器码。
34A5-  C3373E   JP 3E37        ；转到下一程序段。
```

```
3E37-  327D78   LD (787D),A    ；关闭用户中断服务程序出口。
3E3A-  3E10     LD A,10        ；图色码初值（黄色）。
3E3C-  324678   LD (7846),A    ；存入图色控制单元。
3E3F-  C9       RET            ；返回V2.0系统初始化程序。

3FA0-  3AFD68   LD A,(68FD)    ；按住CTRL开机的处理程序；检测键盘第二行。
3FA3-  CB57     BIT 2,A        ；测试是否按下了CTRL键。
3FA5-  3E20     LD A,20        ；A=空格码
3FA7-  2008     JR NZ,3FB1     ；未按CTRL键则跳过八字节。
3FA9-  F640     OR 40          ；使A的D6置1（反白空格码）。
3FAB-  321878   LD (7818),A    ；存入显示状态控制单元I。
3FAE-  321978   LD (7819),A    ；存入显示状态控制单元II。
3FB1-  323C78   LD (783C),A    ；以A为光标所在的字符。
3FB4-  C3C901   JP 01C9        ；转到下一个程序段。

01C9-  3E1C     LD A,1C        ；A=光标复位控制码。

01CB-  CD3A03   CALL 033A      ；光标复位。
01CE-  3E1F     LD A,1F        ；A=清屏控制码。
01D0-  CD3A03   CALL 033A      ；清屏。
01D3-  ED5F     LD A,R         ；取刷新地址寄存器的当前值。
01D5-  32AB78   LD (78AB),A    ；存入随机数种子的中间字节。
01D8-  C9       RET            ；返回。
```

第七章　系统程序的工作

一、系统监控程序

LASER　310 的键盘输入和屏幕显示很有特色，不仅丰富多彩，而且方便灵活。这是它受到用户欢迎的一个重要原因。能实现这一点，除硬件(如 MC6847)的作用外，很大程度上是 V2.0 系统监控程序的贡献。

监控程序是软系统最基础的功能模块之一。简单地说，当你在键盘上按下一个键，例如 CTRL-P，计算机能够知道你输入的是“PRINT”，将它接收下来并且“投射”到屏幕的相应位置，就是监控程序工作的结果。没有监控程序，用户就无法同计算机对话。监控程序是独立于语言处理程序之外的，各种语言程序的输入都可以使用它。

V2.0 的监控程序是以中断服务子程序的形式出现的，与 BASIC 分时、并行，脉冲式地进行工作。它的入口地址固化在内存零页 0038H 的 RST　38H 向量中(JP　2EB8H)。所以 CPU 每次响应 $\overline{\text{INT}}$ 请求，即可自动进入监控程序运行。在执行了关中断指令(DI)后，监控程序将失去作用。下面是监控程序的主模块文本。

```
2EB8-  F5        PUSH AF        ；保护中断断点的现场。
2EB9-  C5        PUSH BC
2EBA-  D5        PUSH DE
2EBB-  E5        PUSH HL
2EBC-  CD7D78    CALL 787D      ；联结用户中断服务子程序的出口。
2EBF-  CD7B3F    CALL 3F7B      ；将显示缓冲区待显示字符送入显示文本区；
                                  根据用户的设定，改变显示状态。
2EC2-  CDDC2E    CALL 2EDC      ；在接受输入的阶段，闪动光标。
2EC5-  CDFD2E    CALL 2EF5      ；扫描键盘一次。
2EC8-  F5        PUSH AF        ；存扫描键盘所得的代码。
2EC9-  213978    LD HL,7839     ；HL=I／O控制字的地址。
2ECC-  CB46      BIT 0,(HL)     ；测试D0（等待输入时D0=0，否则D0=1）。
2ECE-  CC1B30    CALL Z,301B    ；如在输入阶段，将键入字符送入显示文本区。
2ED1-  F1        POP AF         ；取回输入代码。
```

```
2ED2-  CD3034    CALL 3430        ；如按受输入，发声；测试RETURN和BREAK；相
                                   应处理各控制位。
2ED5-  E1        POP HL           ；恢复中断现场。
2ED6-  D1        POP DE
2ED7-  C1        POP BC
2ED8-  F1        POP AF
2ED9-  FB        EI               ；开中断。
2EDA-  EB4D      RETI             ；返回BASIC解释程序的断点。
```

进入监控程序以后，首先是调用用户的中断服务子程序。系统初始化时，在这个出口地址(787DH)已置入一条 RET 指令的代码，将它关闭，使调用立即返回。用户若将自己的监控子程序入口向量放入 787DH～787FH，每次中断就能得到优先执行。

由于要支持简单的硬件实现复杂的功能，监控程序不可避免地具有相当的复杂性，加以它是 V2.0 的原型——Level II 所没有的，研制者尽量利用了原系统中已废弃的部分程序空间，因而程序块零星分散，使用的转移、调用的层次很多，程序的可读性较差。下面对监控程序的剖析，主要讲解各个层次较高的模块，使读者对它的结构和工作原理有一个基本了解。

(一) 显示输出控制

为了避免荧光屏上出现闪烁光斑，BASIC 产生的显示信息不能随时送往显示 RAM 中去。显示字符子程序(CALL 033AH)只负责将字符送到显示缓冲区暂存等待。直到视频系统发来垂直同步信号，进入下边框期，监控程序也开始工作。第一项工作就是把显示缓冲区的内容"趁机"送入显示区。这个程序与后面的有些程序不同，它是无条件执行的，只要中断未被禁止，它都能发挥功能，及时送出显示信息。

V2.0 提供两种屏幕显示状态，由用户向 7818H 置入 0 (浅绿底黑字)或非 0 (深绿底白字)决定。每个字符码送出显示前都要查看这个单元的内容，以确定字符的显示相，例如定义为反相模式下输入的反相字符，就以正相显示。但若用户在工作过程中改变了 7818H 单元的内容，屏幕上的原有字符和以后输入的字符将是反相的，用户和系统都难于区分。所以显示输出程序每次运行都首先测试 7818 的单元内容是否同原来的副本(7819H)一致。若已改变，则将全屏转变为新的显示状态(正相或反相互变)。然后，以显示缓冲区为源数据块，以显示文本区指针(即当前的光标位置)为目标，将显示代码逐一传送，并按代码的不同属性在传送中进行各种处理。

1. 显示信息传送。

```
3F7B-  3A1978    LD A,(7819)    ；取显示状态控制单元II（副本）内容。
3F7E-  47        LD B,A         ；存入B。
3F7F-  2A1878    LD A,(7818)    ；取显示状态控制单元I当前内容。
3F82-  B8        CP B           ；二者相同吗？
3F83-  CAE830    JP Z,30E8      ；相同则不改变屏幕显示状态，转入显示。
3F86-  321978    LD (7819),A    ；将7818H的内容存入7819H，以便下次对照。
3F89-  210070    LD HL,7000     ；改变全屏幕显示状态。HL指向显示区首址。
3F8C-  010002    LD BC,0200     ；设定计数值，改变范围为512个字节。
3F8F-  7E        LD A,(HL)      ；取显示区一个代码。
3F90-  B7        OR A           ；测试。
3F91-  FA973F    JP M,3F97      ；如是图符代码（符号位为1）则不作改变。
3F94-  EE40      XOR 40         ；将字符码的D6取反，从而使其显示反相。
3E96-  77        LD (HL),A      ；将反相字符送回原显示位置。
3F97-  23        INC HL         ；指向显示区下一单元。
3F98-  0B        DEC BC         ；计数减1。
3F99-  78        LD A,B         ；测试BC是否为零（改变完毕）。
3F9A-  B1        OR C
3F9B-  20F2      JR NZ,3F8F     ；未转变完则继续。
3F9D-  C3E830    JP 30E8        ；全屏幕显示状态转变后转入显示。

30E8-  3AAF7A    LD A,(7AAF)    ；取显示缓冲区计数器值。
30EB-  B7        OR A           ；测试是否有待显示的内容。
30EC-  C8        RET Z          ；如缓冲区是空的则返回监控程序。
30ED-  47        LD B,A         ；将缓冲区的字符数存入循环计数器。
30EE-  21B27A    LD HL,7AB2     ；HL=显示缓冲区首址。
30F1-  E5        PUSH HL        ；首址存入堆栈。
30F2-  7E        LD A,(HL)      ；从显示缓冲区取一字节。
30F3-  23        INC HL         ；指向下一单元。
30F4-  E5        PUSH HL        ；存下一地址。
30F5-  C5        PUSH BC        ；存循环计数器值。
30F6-  CD0631    CALL 3106      ；将字符送入显示文本区（是显示控制代码则
                                  进行相应操作）。
30F9-  C1        POP BC         ；取回循环计数值。
30FA-  E1        POP HL         ；取回缓冲区当前地址。
```

```
30FB-  10F5     DJNZ 30F2        ；取下一字节，直到缓冲区内容送完。
30FD-  E1       POP HL           ；取回显示缓冲区首址。
30FE-  22B07A   LD (78B0),HL     ；置为显示缓冲区指针。
3101-  AF       XOR A            ；A清零。
3102-  32AF7A   LD (7AAF),A      ；显示缓冲区计数器清零。
3105-  C9       RET              ；返回监控程序
```

2。显示代码处理。

在 3106H 的子程序中，对传送的字符码进行测试，按字符、图符和显示控制码，分别转到相应的子程序进行处理，然后输出显示。当显示代码是 00H 和 0DH 的控制码时，进行换行操作，将当前行的剩余部分填上空格，光标地址改变为下行的行首地址。

当显示已到屏幕最底行，换行后显示地址将超出 71FFH 的文本区下界时，调用 33F3H 的子程序，将屏上第二行开始的全部显示代码移到显示区最低端(即屏上首行)，并将最底一行全部清为空格，实现了"屏幕卷动"，以便接受新行。

3。屏幕显示的跟踪。

为了给输出的显示码定位，需要一个屏幕显示指针，V2.0 的这个指针就是光标。在系统工作区，有几个关于光标的指针：7820/7821H 是光标当前屏幕位置对应的显示 RAM 地址，78A6H 是光标在本行中的位置，7AAEH 则是光标的下一个位置。系统在清除屏幕时同时使光标指针复位，输出每个字符的同时都要修改光标指针，行结束后显示"READY"后，将光标指针改为下一行首址，按箭号键的实质，也就是通知系统修改光标指针。所以，修改光标指针是显示控制中的一项重要工作。光标指针确定后，光标闪动和字符显示的屏幕位置也就准确无误了。

LASER 310 是全屏幕显示、全屏幕输入、全屏幕编辑，使用非常方便。为了给用户提供这种方便，系统是费了不少"心机"的。因为无论 BASIC 或监控程序，都只能以"行"为处理的数据块。全屏幕共有 16 个显示行，每行 32 个字符。BASIC 的一行程序或直接命令可包含两个显示行，也可只有一个显示行。所以一个显示行可能是单行的程序，也可能是程序行的第一行或第二行这样三种性质。程序输入、编辑的全屏幕操作，都是以程序行为对象的，但光标所能指示的只是显示行，所以必需解决对显示行地位的识别问题。显然，系统是能够识别屏幕每一个显示行的性质的。把光标调入一个双显示行的命令行，无论在上行还是下行按 RETURN 键，都能连同另一行作为整体执行，而在单显示行的命令行中按 RETURN 键，决不会"牵连"相邻的显示行。全屏幕编辑时系统也能准确判断，根据光标所在显示行的性质，或只修改本行，或连同上一行，或连同下一行，决不会搞错。

其中"奥秘"在于，系统在显示每一行时，已将其性质记录在 7AD7H～7AE6H的 16 个单元的数据表中。系统规定：以80H表示单独的一个程序行或非程序的显示文本行，81H 表示双显示行程序中的第一行，00H 表示第二行。每次把光标所在行取到输入缓冲区时，通过查表，根据行性质标志决定是传送 32 字节还是 64 字节，传送首址是本行行首还是上行行首，

末址是本行行末还是下行行末，保证了程序行的完整。在屏幕中间插入新行时，对行性质标志表也要相应调整；屏幕卷动时，要使行性质标志表同步“卷动”；清屏后行性质标志全部置为80H。

显示子程序和输入子程序都要调用33A8H的子程序取行性质标志。它用光标所在行的首址(光标地址-光标的行位置)，计算出以7AD7H为基址的相对地址0～15，将标志取入A。

(二) 键盘检测和键符显示

1. 键盘检测。

本程序在每次响应中断时均无条件调用一次。每次只接受一个按键动作(包括合法的组合按键)，返回时按键的代码在A中，按动功能键的情况反映在有关标志上。

键盘检测的原理已在硬件一章介绍。具体的步骤是：第一，测试整个键盘的输入，迅速判断有无按键动作，有则扫描检测是何键。第二，扫描键盘一周，第二列用作功能控制的各键单独测试。第三，键译码，得到该键代表的字符码。第四，根据功能键置有关标志。第五，对重复按键进行判别和处理。程序如下：

```
2EFD-  3A0068    LD A,(6800)     ；取基本I／0口的输入。
2F00-  F6C0      OR C0           ；将无关位D7和D6置为1，其余位不变。
2F02-  2F        CPL             ；将各位取反，D7，D6肯定为0。
2F03-  FE00      CP 00           ；键盘各列输入的反码是否全为0?
2F05-  2807      JR Z,2F0E       ；若是则未按键，转去置标志后返回。
2F07-  CD282F    CALL 2F28       ；有键按下则进一步扫描测试。
2F0A-  B7        OR A            ；返回时测试A，A=0表示只按了功能键。
2F0B-  C2D705    JP NZ,05D7      ；是字符键进行按键重检后返回监控程序。
2F0E-  213878    LD HL,7838      ；以下处理功能键。HL=键盘状态字地址。
2F11-  CB56      BIT 2,(HL)      ；当前是否处于第四功能输入状态?
2F13-  2808      JR Z,2F1D       ；非第四功能状态则跳过8字节。
2F15-  3A3A78    LD A,(783A)     ；第四功能时测试按键计数器。
2F18-  B7        OR A            ；计数为0吗?（第四功能仅一次按键有效）
2F19-  2802      JR Z,2F1D       ；尚未按过键则保持第四功能标志不变。
2F1B-  CB96      RES 2,(HL)      ；按过键则清除第四功能标志位。
2F1D-  7E        LD A,(HL)       ；取键盘功能状态字。
2F1E-  E606      AND 06          ；保留第四功能和反白输入两标志，其余清0。
2F20-  323878    LD (7838),A     ；存键盘功能状态字。
2F23-  AF        XOR A           ；A清零。
2F24-  323678    LD (7836),A     ；以0作为本次键盘输入代码(无输入)保存。
2F27-  C9        RET             ；返回监控程序。
```

```
2F28-  21FE68   LD HL,68FE    ;键盘扫描。HL=键盘第一行的扫描地址。
2F2B-  0E08     LD C,08       ;设定行计数器=8行。
2F2D-  0606     LD B,06       ;设定列计数器=6列。
2F2F-  7E       LD A,(HL)     ;取本行的输入。
2F30-  F604     OR 04         ;先屏蔽D2（置1），不检测各功能键。
2F32-  1F       RRA           ;A的各位大循环右移，D0移入C标志。
2F33-  302D     JR NC,2F62    ;移入C标志位的是0则是按下的键，转入键译
                                码阶段。
2F35-  10FB     DJNZ 2F32     ;未找到移一位再测试，一行循环检测6次。
2F37-  CB05     RLC L         ;本行无按键，使HL=下一行的扫描地址。
2F39-  0D       DEC C         ;行计数-1。
2F3A-  20F1     JR NZ,2F2D    ;行未扫描完再扫描下一行键。
2F3C-  0604     LD B,04       ;无字符键按下时分别检测第二列的各功能键。
                                扫描计数为4。
2F3E-  21DF68   LD HL,68DF    ;HL=68DFH（第六行）。
2F41-  7E       LD A,(HL)     ;取该行输入。
2F42-  CB57     BIT 2,A       ;测试D2（“-”号键）。
2F44-  2810     JR Z,2F56     ;“-”号键按下则转去译码。
2F46-  CB05     RLC L         ;L小循环左移，HL=68BFH（第七行）。
2F48-  7E       LD A,(HL)     ;取该行输入。
2F49-  CB57     BIT 2,A       ;测试D2（RETURN键）。
2F4B-  280D     JR Z,2F5A     ;RETURN键按下则转去译码。
2F4D-  CB05     RLC L         ;HL=687FH（第八行）。
2F4F-  7E       LD A,(HL)     ;取该行输入。
2F50-  CB57     BIT 2,A       ;测试D2（“:”号键）。
2F52-  280A     JR Z,2F5E     ;“:”号键按下则转去译码。
2F54-  AF       XOR A         ;只可能是单按了CTRL或SHIFT键时，A清零。
2F55-  C9       RET           ;返回。
```

2. 键译码和置键盘标志。

扫描到按下的字符键后，用（6－列计数）×8＋8－行计数的方法，算出一个0～47的相对地址。然后再测试同时按下的功能键，根据键的组合，分别在以下键码表中取得相应目标代

码。每个键码表 48 个字符码，地址是：

01D9H～0208H　第一功能键码表(单按字符键)。

0209H～0238H　第二功能键码表(SHIFT-字符键)。

0239H～0268H　第三功能键码表(CTRL-字符键)。

0289H～02B8H　第四功能键码表(在第四功能输入状态下，CTRL-字符键)。

反白输入字符的键码与以上相同，在送入显示区时再根据键盘功能状态字的反白标志位，将D 6 取反。

按了功能键时，使键盘功能状态字 7838H 有关位置位，各位的意义是：

D0　SHIFT-字符键置 1，检测键盘输入为 0 时复 0 。

D1　CTRL-:键(反白控制)，首次置 1，再次复零，按 RETURN 键亦复 0 。

D2　CTRL-RETURN 置 1，按键后复 0 。

D7　CTRL-字符键置 1，检测键盘输入为 0 时复 0 。

3. 重复按键的检测。

LASER　310 配置的是较为低档的键盘，键的灵活性较差。监控程序以每秒 50 次的速度扫描键盘，键接触时间超过 20ms 就有可能被误检，以为是第二次按键，操作员将难于准确输入程序和命令，对于两次按键作用相反的"反白输入"控制问题尤为突出。所以，每次执行键盘检测程序取得字符码后，并不立即返回，而是转入 05D7H 的子程序进行判别处理。判别的方法比较复杂，并要借助于系统工作区的一些单元。其中，7836H 用来存放连续检测到的键码中的第一键码，7837H 用来存放第二键码，783AH 用在每次键码相同时作为计数器，每检测一次加 1，在连续 42 次内的同一键码均不作为第二次按键。7838H 有关位的作用是：

D3　第一次检测到的键码置 1，键盘输入为 0 时复 0 。

D4　第二次检测到的键码与前者不同时置 1，键盘输入为 0 时复 0 。

D5　两次按键相同，开始重复计数时置 1，计数结束或键盘输入为 0 时复 0 。

D6　重复键标志。一次重复计数结束时置 1，此后的重复计数值减少为 6 次，如仍按键不放，则每检测键盘 6 次即作为另一次按键，输入重复键码。直到松开按键，键盘输入为 0 时，D 6 才复 0 。这就使得键入一串相同字符时比较省事。

4. 键符的显示。

键盘检测程序每次取得一个字符码，既不送入缓冲区，也不显示，仅仅保存在累加器 A 中。如果系统当前并不接受输入，它将不起作用。只有在等待输入阶段，由键符显示子程序将它对应的字符码直接送进显示文本区，待 6847 访问时，立即显示在荧光屏上。V2.0 实际上是利用显示区作为键盘缓冲区，用来容纳输入过程中的程序行，直到行输入结束，从中断返回，再由接受输入子程序将它从显示区传送到输入缓冲区，进行代号化处理。

LASER　310 的一个键码有时需要变成多个字符，所以键符显示并不是简单的传送，还要进行查表翻译。关于"保留词表"详见下一节。程序如下：

```
301B-  B7      OR A          ；测试键码。
301C-  C8      RET Z         ；键码为0则返回。
301D-  F5      PUSH AF       ；键码存入堆栈。
301E-  CD3930  CALL 3039     ；将键码对应字符送显示区，执行控制代码功
                              能，对RETURN和BREAK代码不作处理。
3021-  F1      POP AF        ；取回键码。
3022-  FE0D    CP 0D         ；是RETURN代码吗？
3024-  C8      RET Z         ；是则以零状态返回。
3025-  FE01    CP 01         ；是BREAK代码吗？
3027-  C8      RET Z         ；是则以零状态返回。
3028-  3A3978  LD A,(7839)   ；取I／O控制字内容。
302B-  CB47    BIT 0,A       ；测试是否尚在等待输入阶段（D0=0）。
302D-  C0      RET NZ        ；未等待输入则返回。
302E-  3E20    LD A,20       ；将光标闪烁延时值设定为32。
3030-  324178  LD (7841),A   ；存入计数器使显示字符后的一次闪动较慢。
3033-  2A2078  LD HL,(7820)  ；取当前光标地址。
3036-  C3B23E  JP 3EB2       ；转到下一个程序段。

3039-  213878  LD HL,7838    ；键符显示。HL=键盘功能状态字地址。
303C-  CB7E    BIT 7,(HL)    ；测试是否第三、四功能（CTRL＋）输入码。
303E-  CA5731  JP Z,3157     ；非第三、四功能的键码作单字符显示。
3041-  B7      OR A          ；测试键码的D7。
3042-  F25731  JP P,3157     ；D7=0则非保留词（是→号等），输出。
3045-  F5      PUSH AF       ；是保留词，存代号。
3046-  D680    SUB 80        ；将代号减去80H。
3048-  3C      INC A         ；再加1。
3049-  47      LD B,A        ；存入扫描计数器。
304A-  214F16  LD HL,164F    ；指向保留词表首址之前一字节。
304D-  23      INC HL        ；指向下一字节。
304E-  CB7E    BIT 7,(HL)    ；测试是否保留词首字符变码（D7＝1）。
3050-  28FB    JR Z,304D     ；不是保留词的首字符则测试下一字符。
3052-  10F9    DJNZ 304D     ；是保留词首字符则计数减1，未完则循环。
3054-  7E      LD A,(HL)     ；循环结束HL指向目标保留词首字节，取入A。
3055-  CD8230  CALL 3082     ；显示该字符，指向下一字节。
```

```
3058-  7E       LD A,(HL)      ; 取下一字符。
3059-  CB7F     BIT 7,A        ; 测试是否已是下个保留词。
305B-  28F8     JR Z,3055      ; 仍是本保留词则继续显示其余字符。
305D-  F1       POP AF         ; 取回当前键码。
305E-  0616     LD B,16        ; 为测试后带括号的函数保留词设置计数器。
3060-  219902   LD HL,0299     ; 指向带括号保留词代号表首址。
3063-  BE       CP (HL)        ; 与键码比较。
3064-  2816     JR Z,307C      ; 是带括号保留词之一则显示括号（见下）。
3066-  23       INC HL         ; 不同再指向表的下一字节。
3067-  10FA     DJNZ 3063      ; 表未查完继续查找。
3069-  FEB0     CP B0          ; 非带括号词再测试是否“FN”。
306B-  C0       RET NZ         ; 不是则返回。
306C-  3E20     LD A,20        ; A=空格码。
306E-  CD8230   CALL 3082      ; 显示一个空格。
3071-  3E46     LD A,46        ; A=“F”的代码。
3073-  CD8230   CALL 3082      ; 显示。
3076-  3E4E     LD A,4E        ; A=“N”的代码。
3078-  CD8230   CALL 3082      ; 显示。
307B-  C9       RET            ; 返回。

307C-  3E28     LD A,28        ; 显示括号。A=“(”号代码。
307E-  CD8230   CALL 3082      ; 在保留词后再显示一个正括号。
3081-  C9       RET            ; 返回。

3082-  E67F     AND 7F         ; 屏蔽字符码的D7，使变码复原；其余不变。
3084-  E5       PUSH HL        ; 存当前在保留词表中的扫描地址。
3085-  CD5731   CALL 3157      ; 按显示代码进行输出显示。
3088-  E1       POP HL         ; 恢复保留词表扫描地址。
3089-  23       INC HL         ; 指向下一字节。
308A-  C9       RET            ; 返回。

3EB2-  3A1878   LD A,(7818)    ; 取显示状态控制单元内容。
3EB5-  B7       OR A           ; 测试。
3EB6-  2003     JR NZ,3EBB     ; 在反白显示状态下则转移
```

```
3EB8-  CBB6      RES 6,(HL)       ；将光标所在字符D6复0（正相显示）。
3EBA-  C9        RET              ；返回监控程序。
3EBB-  CBF6      SET 6,(HL)       ；在反白状态下使光标所在字符呈反白显示。
……
```

本程序调用的子程序3157H，在前面的显示输出子程序中也曾调用(经3106H的子程序)。它是一个负责"分拣发送"的程序段，对A中的显示代码进行测试后，按其性质(图符、字符或显示控制码)送往相应的子程序去处理和输出。

（三） 辅助功能

1. 光标闪动。

LASER 310在等待输入时，屏幕的下一个显示位置上缓缓闪动着一个"光标"，给用户作输入位置指示。粗看起来它是一个时隐时现的黑方块。仔细观察便能发现，实际上是那个位置的字符在反复地改变显示状态(正相与反相互变)。监控程序每秒可执行50次，如果每次都变反，由于人眼的视觉滞留效应将看不出变化，所以再加以延时，执行本程序16次才闪动一次，周期约为1/3秒。系统不等待输入时，响应中断后闪光标程序是不执行的。程序文本如下：

```
2EDC-  3A3978    LD A,(7839)      ；取I/O控制字。
2EDF-  CB47      BIT 0,A          ；测试D0（系统是否在等待输入）。
2EE1-  C0        RET NZ           ；控制位非0(不接受输入)则返回，不闪光标。
2EE2-  214178    LD HL,7841       ；HL＝光标闪烁延时计数器地址。
2EE5-  35        DEC (HL)         ；计数减1。
2EE6-  C0        RET NZ           ；计数值不为0(延时未满)不闪动，返回。
2EE7-  3E10      LD A,10          ；延时已到，重设延时初值。
2EE9-  324178    LD (7841),A      ；存入计数器。
2EEC-  2A2078    LD HL,(7820)     ；取当前光标所在地址。
2EEF-  3E40      LD A,40          ；令A=40H（01000000B）。
2EF1-  AE        XOR (HL)         ；使光标所在字符的D6变为原来的反码，从而
                                   使字符显示状态变反。
2EF2-  77        LD (HL),A        ；放入原来位置。
2EF3-  C9        RET              ；返回监控程序。
```

2. 键鸣和输入结束控制。

这个子程序的主要内容也是在等待输入的阶段才执行的。它除产生按键的鸣叫声以表示按键有效外，另一重要功能是检查RETURN和BREAK键，如按了其中之一，就把有关标志置位，表示输入行已经结束。BASIC的接受输入子程序发现后，就会结束等待键盘输入状态，

将输入的一行送进缓冲区，或进入 BREAK 状态。程序如下：

```
3430-  213978   LD HL,7839     ；HL=I／O控制字地址。
3433-  B7       OR A           ；测试键码。
3434-  200B     JR NZ,3441     ；键码为0则只延时不发声，键码非0则跳转。
3436-  CBCE     SET 1,(HL)     ；将控制字D1置1表示无输入。
3438-  01FF03   LD BC,03FF     ；设定延时计数值。
343B-  0B       DEC BC         ；计数减1。
343C-  79       LD A,C         ；测试计数值是否到0。
343D-  B0       OR B
343E-  20FB     JR NZ,343B     ；延时未满则继续。
3440-  C9       RET            ；返回监控程序。
3441-  CB46     BIT 0,(HL)     ；测试是否在等待输入。
3443-  C0       RET NZ         ；不接受输入则返回。
3444-  FE0D     CP 0D          ；测试是否是RETURN键码。
3446-  2806     JR Z,344E      ；是则转去置标志。
3448-  FE01     CP 01          ；测试是不是BREAK键码。
344A-  2004     JR NZ,3450     ；不是则跳过置标志指令。
344C-  CBD6     SET 2,(HL)     ；D2置1，表示按了BREAK键。
344E-  CBC6     SET 0,(HL)     ；D0置1，表示输入行结束(以RETURN或BREAK)。
3450-  E5       PUSH HL        ；存I／O控制字地址。
3451-  21A000   LD HL,00A0     ；设定发声时间控制值。
3454-  010600   LD BC,0006     ；设定发声频率控制值。
3457-  CD5C34   CALL 345C      ；发出蜂鸣声。
345A-  E1       POP HL         ；取回I／O控制字地址。
345B-  C9       RET            ；返回监控程序。
```

二、BASIC 输入程序

BASIC 输入程序是支持用户用键盘输入 BASIC 程序的功能模块。它把用户键入的由监控程序接收的一行字符，从显示区取到输入缓冲区，并整理成标准的 BASIC 格式存放到用户程序区。如果输入的是无行号的直接执行语句，便由解释程序在输入缓冲区解释执行。

系统初始化以后，或一条命令、一个程序执行完毕时，屏幕上显示“READY”提示符，光标在下一行首缓缓闪动，等待输入新的程序或命令。这时，系统就是在 BASIC 输入程序中运行。接受一行语句或命令，只使用输入程序一次。按 RETURN 键以前，系统仅是在几个字节

间循环等待，直到输入行结束才继续往下执行。

V2.0的全屏幕编辑功能，也是利用BASIC输入程序实现的。因为编辑的过程也就是输入的过程，在某行中按RETURN键，等于将现在屏幕上的本行内容重新输入，自然就取代了内存中行号相同的原有行。

BASIC输入程序的主模块地址在1A19H～1AF7H。它的工作流程是：第一步，置输出设备为显示器，显示"READY"并将光标显示在下一行开始位置；第二步，根据用户定义自动产生行号(V2.0未用此功能)；第三步，调用接受输入子程序，等待和接受输入行；第四步，调用代号化子程序，将输入的行变成标准BASIC格式；第五步，如输入语句无行号则转入执行阶段，如有行号就将它移入用户区保存，并相应调整程序区内容及有关指针；第六步，转回第二步，接受下一个输入行。

本程序的前部，是自动产生行号程序段。这是Level II系统原有的，用AUTO命令及有关行号参数，便可由系统自动产生行号序列，用户只需键入行的内容。若自动产生的行号在内存里已有同号的程序行，则将该行显示出来，由用户决定取舍。V2.0既未使用此功能，也未删除这一段程序，只是在系统初始化时把作"AUTO标志"的78E1H单元置为0，使本程序段不工作。用户若用POKE手段将起始行号装入78E2/78E3H，行号增量装入78E4/78E5H，最后再使78E1H为非零，便能使用AUTO功能。输入中按BREAK键可中止AUTO功能。

(一) 主模块

```
1A19-  CD8B03    CALL 038B         ；置当前输出设备为显示器。
1A1C-  CDAC79    CALL 79AC         ；DOS 出口。
  ⋮                                ；三字节未用。
1A22-  CDF920    CALL 20F9         ；换行。
1A25-  212919    LD HL,1929        ；指向存有"READY"字符串的首址。
1A28-  CDA728    CALL 28A7         ；调用显示字符串子程序显示"READY"。
  ⋮                                ；以下八字节不用。
1A33-  21FFFF    LD HL,FFFF        ；HL=-1。
1A36-  22A278    LD (78A2),HL      ；使当前行号指针为 -1，表示尚在输入阶段。
1A39-  3AE178    LD A,(78E1)       ；取 AUTO标志。
1A3C-  B7        OR A              ；测试(非零——自动产生行号)。
1A3D-  283A      JR Z,1A79         ；如非自动设行方式则跳过以下程序段。
1A3F-  2AE278    LD HL,(78E2)      ；取自动产生的当前行号(第一次为设定的初
                                     值，以后为上一次执行本程序段的结果)。
1A42-  E5        PUSH HL           ；同时将当前行号存入堆栈。
1A43-  CDAF0F    CALL 0FAF         ；将该行号值变为ASCII字符显示于行首。
1A46-  3E20      LD A,20           ；令A=空格的 ASCII码。
1A48-  CD2A03    CALL 032A         ；在行号后显示一空格。
```

```
1A4B-  D1       POP DE         ；将AUTO的当前行号取入DE。
1A4C-  D5       PUSH DE        ；再存备用。
1A4D-  CD2C1B   CALL 1B2C      ；在程序区检索有否此行号的程序行。
1A50-  DC532E   CALL C,2E53    ；如已有同行号程序则显示该行(光标置行末)。
1A53-  00       NOP            ；此字节未用。
1A54-  CDE303   CALL 03E3      ；无相同行号则等待并接受输入行(若输入过程
                                 中按了BREAK键，返回时C标志置位)。
1A57-  D1       POP DE         ；行输入结束，再把当前行号取入 DE。
1A58-  3006     JR NC,1A60     ；如果没有按BREAK键则跳过6字节。
1A5A-  AF       XOR A          ；按了BREAK键则把 A清零。
1A5B-  32E178   LD (78E1),A    ；清除AUTO标志。
1A5E-  18B9     JR 1A19        ；结束自动设行，转回本程序入口接受新行。
1A60-  2AE478   LD HL,(78E4)   ；取用户设定的行增量值。
1A63-  19       ADD HL,DE      ；当前行号与增量相加，看最高位是否进位。
1A64-  38F4     JR C,1A5A      ；下一个行号超过65535（C标志置位）则结束
                                 自动产生行号的过程。
1A66-  D5       PUSH DE        ；再存当前行号于堆栈。
1A67-  11F9FF   LD DE,FFF9     ；令DE＝V2.0允许的最大行号。
1A6A-  DF       RST 18         ；与下一个行号（在HL中）比较。
1A6B-  D1       POP DE         ；取回当前行号。
1A6C-  30EC     JR NC,1A5A     ；如下一个行号大于允许值也结束AUTO。
1A6E-  22E278   LD (78E2),HL   ；保存下个行号备用。
  ⋮                            ；二字节未用。
1A73-  21E779   LD HL,79E7     ；指向输入缓冲区的前一字节。
1A76-  C3811A   JP 1A81        ；转入对输入行的处理。
1A79-  00       NOP            ；以下二字节未用。
1A7A-  00       NOP
1A7B-  CDE303   CALL 03E3      ；等待并接受输入行，行输入结束返回时HL＝
                                 缓冲区前一字节地址（79E7H）。
1A7E-  DA331A   JP C,1A33      ；如按了BREAK键则转去重新接受输入行。
1A81-  D7       RST 10         ；将HL所指地址的下一单元内容取入 A（跳过
                                 空格），如是数字(行号)则C标志置位）。
1A82-  3C       INC A          ；这是不影响C标志测试A是否为零的手段。
```

```
1A83-   3D        DEC A            ；如Z标志置位则A=0
1A84-   CA331A    JP Z,1A33        ；是空语句(缓冲区首字节为0)则去接受下行。
1A87-   F5        PUSH AF          ；保存检测首字符是否数字的状态标志(C)。
1A88-   CD5A1E    CALL 1E5A        ；把 ASCII码的行号变成二进制值放入DE，
                                     执行后HL指向语句体第一个非空格字符。
1A8B-   2B        DEC HL           ；从语句体首字符后退一字节。
1A8C-   7E        LD A,(HL)        ；将所指单元内容取入 A。
1A8D-   FE20      CP 20            ；它是空格吗？
1A8F-   28FA      JR Z,1A8B        ；如是再退，直到行号末尾或缓冲区前一字节。
1A91-   23        INC HL           ；指向下一字节。
1A92-   7E        LD A,(HL)        ；取行号后（或缓冲区的）第一个字符。
1A93-   FE20      CP 20            ；它是空格吗？
1A95-   CCC909    CALL Z,09C9      ；行号后的一个空格略去不存（今后LIST会自
                                     动在行号后显示一空格）。
1A98-   D5        PUSH DE          ；行号值存入堆栈。
1A99-   CDC01B    CALL 1BC0        ；输入语句体代号化，返回时BC=语句字节数。
1A9C-   D1        POP DE           ；行号值取入DE。
1A9D-   F1        POP AF           ；取回测试有无行号时得到的C标志状态。
1A9E-   22E678    LD (78E6),HL     ；将代号化语句首址存入指针单元。
1AA1-   CDB279    CALL 79B2        ；DOS出口。
1AA4-   D25A1D    JP NC,1D5A       ；如语句无行号（直接命令）则转入解释执行。
1AA7-   D5        PUSH DE          ；存行号值。
1AA8-   C5        PUSH BC          ；存代号化语句长度。
1AA9-   AF        XOR A            ；A清零。
1AAA-   32DD78    LD (78DD),A      ；设置输入阶段的标志。
1AAD-   D7        RST 10           ；取语句体第一个字符。
1AAE-   B7        OR A             ；检测行号后是否为语句结尾符（删行命令）。
1AAF-   F5        PUSH AF          ；存检测结果的状态标志（Z）。
1AB0-   EB        EX DE,HL         ；行号值换入HL。
1AB1-   22EC78    LD (78EC),HL     ；行号存入指针单元。
1AB4-   EB        EX DE,HL         ；行号换入DE，供检索行号子程序使用。
1AB5-   CD2C1B    CALL 1B2C        ；在程序区检索相同行号。执行后BC为新行应
                                     安放的首地址（有相同的行号，BC=该行的首
                                     址，否则BC=大于该行号的下行程序首址；如
```

```
                                      原有行号均小于该行号，则BC=最后语句的结
                                      束符后的地址）。
1AB8-  C5       PUSH BC              ；BC值存入堆栈。
1AB9-  DCE42B   CALL C,2BE4          ；如找到相同行号，则将下一行及以后所有程
                                      序行向前移，复盖掉该行内容（删除）。
1ABC-  D1       POP DE               ；DE=安放新行的首址。
1ABD-  F1       POP AF               ；取回检测行号后边有无内容的 Z标志状态。
1ABE-  D5       PUSH DE              ；重新存入新行安放地址。
1ABF-  2827     JR Z,1AE8            ；如行号后为00H（相同行号的行已在1AB9H时
                                      删去）则转去修改行指针。
1AC1-  D1       POP DE               ；取回新行安放地址。
1AC2-  2AF978   LD HL,78F9           ；HL=程序终址。
1AC5-  E3       EX (SP),HL           ；HL=新行的长度，程序终址存入堆栈。
1AC6-  C1       POP BC               ；程序终址弹出到BC。
1AC7-  09       ADD HL,BC            ；HL=加上新行的字节数后新的程序终址值。
1AC8-  E5       PUSH HL              ；保存新的终址。
1AC9-  CD5519   CALL 1955            ；测试空闲区是否够用，然后把新行安放地址
                                      后的内容向高端传送，腾出放新行的空间。
1ACC-  E1       POP HL               ；HL=程序新的终址。
1ACD-  22F978   LD (78F9),HL         ；新终址存为指针。
1AD0-  EB       EX DE,HL             ；HL=安放新行的首址。
1AD1-  74       LD (HL),H            ；在行的指针位置预置一个非零字节，避免两
                                      字节都为零被以后子程序误认为程序结束。
1AD2-  D1       POP DE               ；DE=新行行号值。
1AD3-  E5       PUSH HL              ；存新行首址。
1AD4-  23       INC HL               ；指向行的第三字节。
1AD5-  23       INC HL
1AD6-  73       LD (HL),E            ；将行号值的低八位放入。
1AD7-  23       INC HL               ；指向第四字节。
1AD8-  72       LD (HL),D            ；将行号值的高八位放入。
1AD9-  23       INC HL               ；指向下一字节。
1ADA-  EB       EX DE,HL             ；DE=存放语句内容的首址。
1ADB-  2AA778   LD HL,(78A7)         ；取输入缓冲区首址。
```

```
1ADE-  EB       EX DE,HL        ; DE=缓冲区首址，HL=语句体存放首址。
1ADF-  1B       DEC DE          ; 后退两字节到存放代号化语句的首址。
1AE0-  1B       DEC DE
1AE1-  1A       LD A,(DE)       ; 从缓冲区取一字节。
1AE2-  77       LD (HL),A       ; 放入程序区相应位置。
1AE3-  23       INC HL          ; 指向下一个存放单元地址。
1AE4-  13       INC DE          ; 指向下一个应取字节地址。
1AE5-  B7       OR A            ; 测试刚才传送的是否00H。
1AE6-  20F9     JR NZ,1AE1      ; 如非语句结尾符则转去传送下一字节。
1AE8-  D1       POP DE          ; 取回刚传送行的首地址（行指针地址）。
1AE9-  CDFC1A   CALL 1AFC       ; 修改从新行开始的所有行指针。
1AEC-  CDB579   CALL 79B5       ; DOS出口。
1AEF-  CD5D1B   CALL 1B5D       ; 使 BASIC各指针初始化。
1AF2-  CDB879   CALL 79B8       ; DOS出口。
1AF5-  C3331A   JP 1A33         ; 循环，输入下一行（不再显示“READY”）。
```

（二） 子程序

BASIC输入程序调用的子程序中，有的是通用子程序，如“取下一字符”、“检索行号”和“修改行指针”等，参见“子程序”一节。下面着重介绍“接受输入”子程序和“代号化”子程序。

1. 接受输入子程序。

（1）程序入口。03E3H(程序中转移很多，不便给出程序块的地址范围)。

（2）入口条件。由BASIC输入程序调用时7839H的D4＝0；INPUT子程序调用时D4＝1。

（3）程序内容。

①等待输入行。其中有以下程序段：

```
   ⋮
0405-  213978   LD HL,7839      ; HL＝I／O控制字地址。
0408-  CB86     RES 0,(HL)      ; 将其D0复位为0，进入等待输入状态。
040A-  CB96     RES 2,(HL)      ; 将其D2复位为0（将由BREAK置为1）。
040C-  CB46     BIT 0,(HL)      ; 测试7839H的D0。
040E-  28FC     JR Z,040C       ; 如D0为0则再行测试。
   ⋮
```

单从这里来看，既然前面已使D0为0，无论怎么测试肯定还是0，040CH～040FH之间无疑形成一个死循环，程序将不能再向下运行。但是，系统现处于允许中断的状态，所以循环将不会持续到20ms以上，一旦CPU响应$\overline{INT}$请求，这一循环就暂时被打断，监控程序开始

工作。由于7839H的等待输入标志位D0为0，监控程序将把键盘输入的字符送入显示文本区，并使蜂鸣器发出短暂的叫声，然后从中断返回。于是，040CH～040FH间的循环再度开始，直到下一次中断。只有在操作者按了RETURN或BREAK键时，监控程序将7839H的D0置位为1（BREAK并且将D2置1），这次从中断返回，040CH的测试指令将得到非零状态，于是脱出循环，程序得以向下运行。

②测试输入行的规模(一个显示行还是两个显示行，当前光标在上行还是下行)，以确定传输32字节还是64字节，以及传输的源首址。

③将显示区的当前输入行，逐字符送入输入缓冲区。如用户设定为反白显示状态，则将反白字符恢复成正常字符送入。传送时检测引号，图案符号和反相字符如出现在引号外，则显示“语法错误”信息，并转回1A19H入口，重新接受输入。

④传送完毕逆行跳过行后部的所有空格，在最末一个字符后的空格处放入结尾符00H。换行并终止反白输入状态。如果是BREAK结束输入行，令A=01H，C标志置1，返回。

(4)出口状态。按BREAK键返回时C标志置1，BASIC输入程序调用后，HL指向输入缓冲区首址前一字节，INPUT子程序调用后，HL指向缓冲区中用户键入数据首址前一字节。

2. 代号化子程序。

(1)程序地址。1BC0H～1B8FH。

(2)入口条件。HL指向输入缓冲区内ASCII代码串(即直接命令的首字节，或程序语句行号以后的地址)。所有寄存器均被使用。

(3)程序内容。以HL为指针逐字节扫描ASCII代码串，对每个遇到的字符，分别不同情况，将对应BASIC语法元素依次存入代号化缓冲区(从输入缓冲区首址之前两个单元开始，逐渐向后覆盖输入代码串已处理部分)。方法是：

①对于＞2FH并＜3CH的代码(即数字、“:”号、“;”号的ASCII码)，原样送入代号化缓冲区。

②如进入DATA、REM(包括与REM等效的“'”号)语句，就将整个语句的代码串原样传送入代号化缓冲区。

③遇到第一个引号后，将它及以后的字符串原样送入代号语句缓冲区，直至扫描到第二个引号或语句结束为止；

④上面两种情况之外的“?”号用PRINT代号(B2H)取代，“'”号用冒号连同REM代号(3AH和93H)取代，送入代号化缓冲区。

⑤除上述各种情况以外，遇到＜30H或＞3BH的代码时，就到“保留词表”中去检索。在表中找不到与之相同的保留词，说明它是变量名(或是语句中的错误)，将它们原样送入代号化缓冲区；如查出它们与某一个保留词相同，检索时扫描过的保留词计数+7FH，即是它的代号，把代号送入代号语句缓冲区，若是ELSE代号，还要在前面放入一个冒号码。

⑥遇到多语句行中的语句分隔符“:”号(REM语句中的除外)，重新进行上述全过程，扫描下个语句。

⑦遇到行结尾符——00H，结束代号化过程，返回BASIC输入程序。

(4)出口状态。A=0，BC=代号化代码串长度+5(加上指针、行号和结尾符后，语句

应有的总长度)， DE = 代号化代码串后三个零字节最后一个的地址，HL = 代号化代码串首址 - 1。

3. 保留词表。

保留词是系统用于语句、命令、函数及算符的专门单词，为避免混淆禁止用户作为变量名。保留词表是 V2.0 系统的基本表格之一。它将系统全部命令、语句定义符和函数、运算符号的保留词的 ASCII 字符码按代号大小顺序排列，地址在 1650H～1821H 。从 END (80H)开始。词之间不加空格和标志。为便于识别，每个词第一个字符作了“变码”处理，将本来代码的符号位(D 7)置为 1 (= 原码 + 80H)，变成负数形式。例如 END 的三个 ASCII 码是 45H，4 EH，44H，而在表中成了 C54E44 。检索保留词表时，以寄存器 B 为计数器，令其初值 = 7 FH，每扫描到一个符号为负的代码则 B + 1 。这样，当找到相同的保留词时，B 中就是它的代号了。

可供使用的代号共 123 个，V2.0 系统未全部使用。未用代号在保留词表的相应位置上，有若干零字节(首字节亦变码为 80H)，以便扫描时产生正确的代号计数。这种情况是对 Level II 系统进行修改所造成的。V2.0 系统的保留词及代号见下表：

表 7-1

END	80H	FOR	81H	RESET	82H	SET	83H
CLS	84H	NEXT	87H	DATA	88H	INPUT	89H
DIM	8AH	READ	8BH	LET	8CH	GOTO	8DH
RUN	8EH	IF	8FH	RESTORE	90H	GOSUB	91H
RETURN	92H	REM	93H	STOP	94H	ELSE	95H
COPY	96H	COLOR	97H	VERIFY	98H	CRUN	9CH
MODE	9DH	SOUND	9EH	OUT	A0H	LPRINT	AFH
POKE	B1H	PRINT	B2H	CONT	B3H	LIST	B4H
LLIST	B5H	CLEAR	B8H	CLOAD	B9H	CSAVE	BAH
NEW	BBH	TAB(	BCH	TO	BDH	USING	BFH
USR	C1H	POINT	C6H	INKEY$	C9H	THEN	CAH
NOT	CBH	STEP	CCH	+	CDH	-	CEH
×	CFH	/	D0H	↑	D1H	AND	D2H
OR	D3H	>	D4H	=	D5H	<	D6H
SGN	D7H	INT	D8H	ABS	D9H	INP	DBH
SQR	DDH	RND	DEH	LOG	DFH	EXP	E0H
COS	E1H	SIN	E2H	TAN	E3H	ATN	E4H
PEEK	E5H	LEN	F3H	STR$	F4H	VAL	F5H
ASC	F6H	CHR$	F7H	LEFT$	F8H	RIGHT$	F9H
MID$	FAH						

三、执行驱动程序

BASIC 语言的最突出优点之一就是它接近自然语言，易为初学者接受。但 BASIC 的语句和程序，与 Z80 的机器指令和程序是完全不同的，因而 CPU 并不能直接执行。系统程序的一种重要模块，就是在用户的高级语言程序和 CPU 指令系统之间担任"翻译"的语言处理程序。它能够根据源程序的内容要求，用相应的机器语言程序来加以表达和实现。语言处理程序一般可分为两类。一类是"编译程序"。它相当于"笔译"方式，是把整个源程序完整地译成等效的机器语言目标程序。然后，"原文"和"翻译"都可退场，目标程序可以独立地存贮和运行。CPU 直接执行"译文"，就能完成源程序所规定的任务。另一种是"解释程序"。它相当于"口译"方式，系统逐词逐句地"阅读"源程序，在每一语法单位读过后立即调用相应的机器语言程序进行表达。这样，源程序读完也就翻译并执行完毕，并不留下"译文"，执行时需要"原文"和"翻译"并存，每次都是重新进行。二者比较，编译方式显然比解释方式节省内存，而且执行速度快。但解释方式比较灵活，便于对源程序调试和修改，在运行中不断充实完善，因而较适合初学者。BASIC 主要就是为初学者研制的，所以它的语言处理程序虽也有编译方式的版本，但多数采用解释方式。V2.0 的语言处理程序就是一种 BASIC 解释程序。

（一） "执行"和"驱动"

在 BASIC 级所说的"执行"程序，从系统的角度看就是对 BASIC 程序的"解释"。解释工作是动态的，在对程序语句表从头到尾扫描的过程中进行。因此，我们把系统中负责推动基本扫描过程的模块称为"执行驱动程序"。

V2.0的执行驱动程序地址在 1D1EH～1D90H。它以语句为扫描和处理的最小单元，负责处理语句间的链结构元素，取入语句命令代号，然后根据它转入特定的工作子程序。语句命令工作子程序则负责扫描语句的其余部分，以每个语法元素为单元进行处理，包括必要时调用有关的功能子程序。一个工作子程序执行结束后，又应回到执行驱动程序，开始扫描下一个语句。如此交替进行，直至程序结束。

为了实现这种螺旋式前进的运行方式，系统设计的主要技巧有二：

第一，规定以 HL 寄存器对作为程序扫描执行的地址指针。进入执行驱动程序时，就使 HL 指向程序或命令之前一个字节的地址，从此处开始扫描执行。在执行程序的全过程中，HL 必须随着对程序的逐字节扫描同步增值，随时指向当前到达的地址。从执行驱动程序到工作子程序到各种功能子程序，交接时必须保证 HL 指针的准确性，才能使层次纷繁的各级程序模块的工作准确衔接，使解释执行"一字不重，一字不漏"。因此，当需要 HL 寄存器作其他用途时，必须将它的扫描指针值压入堆栈，换入别的空闲寄存器或放到系统工作区保存起来，并在恢复扫描进程前及时取回。

第二，执行驱动程序在进入每一语句之初，先往堆栈压入自己的入口地址——1D1EH，然后转移到(而不是调用)该语句的工作子程序。工作子程序执行完毕时，已扫描到语句的末尾，HL指向下一个语句的前一字节(00H 或:号码)，工作子程序在堆栈里已无残留数据，栈顶就是1D1EH，所以最后的 RET 指令便会使它弹入程序计数器 PC，重新进入执行驱动程序，解释执行下一个语句。这样，就能保持连锁运行直至发现链指针是 0000H 为止。

（二） 工作子程序地址表

执行驱动程序在取入语句命令代号后就访问存放在1822H～1899H的工作子程序地址表，以取得入口地址。函数等代号的子程序地址，由表达式求值程序调用。TAB、TO、USING、THEN和STEP的代号，总是在特定语句中使用，所以是作为那些工作子程序的组成部分。

表7-2是语句命令工作子程序地址表，其中包括一些V2.0未使用的代号子程序地址和磁盘BASIC代号的转移地址。在表中分别用下划线和"#"号注明。

表 7-2

代号	地址	代号	地址	代号	地址
80H (END) -	1DAEH	95H (ELSE) -	1F07H	AAH (#) -	7991H
81H (FOR) -	1CA1H	96H (COPY) -	3912H	ABH (#) -	7997H
82H (RESET) -	0138H	97H (COLOR) -	389DH	ACH (#) -	799AH
83H (SET) -	0135H	98H(VERIFY)	3738H	ADH (#) -	79A0H
84H (CLS) -	01C9H	99H(DEFINT) -	1E03H	AEH(SYSTEM) -	0000H
85H (#) -	7973H	9AH(DEFSNG) -	1E06H	AFH(LPRINT) -	2067H
86H(RANDOM) -	01D3H	9BH(DEFDBL) -	1E09H	B0H (#) -	795BH
87H (NEXT) -	22B6H	9CH (CRUN) -	372EH	B1H(POKE) -	2CB1H
88H (DATA) -	1F05H	9DH (MODE) -	2E63H	B2H(PRINT) -	206FH
89H (INPUT) -	219AH	9EH (SOUND) -	2BF5H	B3H(CONT) -	1DE4H
8AH (DIM) -	2608H	9FH(RESUME) -	1FAFH	B4H(LIST) -	2B2EH
8BH (READ) -	21EFH	A0H (OUT) -	2AFBH	B5H(LLIST) -	2B29H
8CH (LET) -	1F21H	A1H (ON) -	1F6CH	B6H(DELETE) -	2BC6H
8DH (GOTO) -	1EC2H	A2H (#) -	7979H	B7H (AUTO) -	2008H
8EH (RUN) -	1EA3H	A3H (#) -	797CH	B8H(CLEAR) -	1E7AH
8FH (IF) -	2039H	A4H (#) -	797FH	B9H(CLOAD) -	3656H
90H(RESTORE) -	1D91H	A5H (#) -	7982H	BAH(CSAVE) -	34A9H
91H(GOSUB) -	1EB1H	A6H (#) -	7985H	BBH(NEW) -	1B49H
92H(RETURN) -	1EDEH	A7H (#) -	7988H		
93H (REM) -	1F07H	A8H (#) -	798BH		
94H(STOP) -	1DA9H	A9H (#) -	798EH		

为了使读者更容易理解，我们结合一个实例说明执行驱动程序的工作过程。假定用户程序区有以下程序：

```
10  PRINT  A
20  END
```

程序输入后程序区内容如下：

7AE9	7AEA	7AEB	7AEC	7AED	7AEE	7AEF	7AFO	7AF1	7AF2
F1	7A	0A	00	B2	20	41	00	F7	7A

7AF3	7AF4	7AF5	7AF6	7AF7	7AF8
14	00	80	00	00	00

我们从键盘发出命令 RUN，机器就会“不假思索”地“立即”显示出结果（0）。然而实际上并不那么简单。下面让我们看看执行的全过程。

（三） 语句体的执行驱动

执行驱动程序分为两个程序段，1D1EH～1D59H 用来扫描处理链结构元素，1D5AH 开始对语句体的执行驱动。1D1EH 和 1D5AH 各成为一个入口。要解释执行用户程序区的 BASIC 程序，必须先执行一条直接命令（一般为 RUN 或 GOTO）。这些命令的工作子程序会把程序执行指针指向目标程序的前一字节，然后再转入执行驱动程序 1D1EH 入口，从链指针开始扫描。而用户从键盘发出的不带行号的直接命令，在 BASIC 输入程序后部，便将 HL 指向了代号化缓冲区前一字节，直接转入 1D5AH。程序行中间“:”号后的语句，也由这个入口对语句体开始扫描。

语句体的首字节，按 BASIC 语法，除省略 LET 的赋值语句以字母开始之外都应是语句定义符的代号。执行驱动程序最重要的任务，就是取语句代号并查出它的工作子程序地址。因为工作子程序地址表是按代号值排列，每个地址占两字节，所以（代号－80H）×2＋1822H 便是目标地址低字节的存放单元。

在转入工作子程序前，先将代号后的第一个非空格字符取入累加器 A 并进行测试。这就是 V2.0 系统程序使用非常多的“取下一字符”子程序（入口 1D78H，用 RST10H 指令调用）。它能跳过空格码 20H，根据扫描到的第一个字符是数字、其他字符或语句结束符，以标志状态表示。很多工作子程序需要据此判断程序是否正确或进行不同的处理。

在等待输入的状态下键入 RUN，然后按 RETURN 键，系统从监控程序返回 BASIC 输入程序，从 1A7EH 继续执行。在 1A99H 使 RUN 代号化为 8EH。代号化代码串存放在 79E4H～79EAH，内容是8E000000。HL 指向它们的前一字节 79E5H（系统初始化时已在其中存放“:”号码）。执行到 1AA4H，因经测试无行号，就转入执行驱动程序的 1D5AH 入口。因此，我们从这里开始剖析。在系统程序反汇编文本的注释中，本实例在执行中的当前值和状态，用方括号表示，不再注明。程序如下：

```
1D5A-  D7        RST 10       ；取语句体首字符（详见1D78H的RST 10程序），
                               [A=8E,Z标志复位，HL=79E5H]。
1D5B-  111E1D    LD DE,1D1E   ；将执行驱动程序的入口地址压入堆栈，以备
1D5E-  D5        PUSH DE       工作子程序执行后继续解释执行下个语句。
1D5F-  C8        RET Z        ；如句首是零或：号（空语句）则返回（1D1EH）。
1D60-  D680      SUB 80       ；首字节代码值-80H，如是代号则得出它在工
                               作子程序表中的位置[A=0EH,第14个地址]。
```

```
1D62-  DA211F   JP C,1F21     ；如首字节<80H可能是省去LET定义符的赋值语
                                句，转LET子程序；如为错误亦由它处理。
1D65-  FE3C     CP 3C         ；如代号>=BCH(80H+3CH),则非语句命令代号
1D67-  D2E72A   JP NC,2AE7    ；转去作进一步测试和处理。
1D6A-  07       RLCA          ；代号在工作子程序表的位置×2[A=1CH]。
1D6B-  4F       LD C,A        ；将它作为在子程序中的相对地址存入BC.[相
1D6C-  0600     LD B,00       ；对地址在表的第28单元]
1D6E-  EB       EX DE,HL      ；为了使用HL先把程序执行指针值存入DE。
1D6F-  212218   LD HL,1822    ；令HL=工作子程序表首址。
1D72-  09       ADD HL,BC     ；求出该代号子程序地址的存放地址[183EH]。
1D73-  4E       LD C,(HL)     ；将子程序地址低字节取入C[C=A3H]。
1D74-  23       INC HL        ；指向高字节存放地址[183FH]。
1D75-  46       LD B,(HL)     ；取入B，BC=工作子程序地址[1E3AH]。
1D76-  C5       PUSH BC       ；将工作子程序地址压入堆栈，下面执行RET指
                                令便可转入这个子程序执行。
1D77-  EB       EX DE,HL      ；恢复程序执行指针[HL=79E5H]。
1D78-  23       INC HL        ；HL指向下一个字节；以下程序段同时是RST
                                10H（取下个字符）工作程序。
1D79-  7E       LD A,(HL)     ；将下个字节内容取入A。[A=00H]
1D7A-  FE3A     CP 3A         ；与冒号码相比（是冒号则 Z标志置位）。
1D7C-  D0       RET NC        ；如是冒号、字母、符号则返回调用的程序，
                                在执行驱动程序中是借此转入工作子程序。
1D7D-  FE20     CP 20         ；测试它是否空格。
1D7F-  CA781D   JP Z,1D78     ；如是空格则跳过，再取下一字节。
1D82-  FE0B     CP 0B         ；测试控制代码(对V2.0无意义)，与0BH比较。
1D84-  3005     JR NC,1D8B    ；如不是<=0BH的零或控制码则转移。
1D86-  FE09     CP 09         ；再同09H比较。
1D88-  D2781D   JP NC,1D78    ；如是控制码09H和0AH则跳过，取下一字节。
1D8B-  FE30     CP 30H        ；与0的ASCII码比较，是数字C复位否则置位。
1D8D-  3F       CCF           ；C标志取反（数字-C,符号或零-NC）。
1D8E-  3C       INC A         ；测试是否零字节（不影响C标志）。
1D8F-  3D       DEC A         ；如是零则Z标志置位[Z、NC]。
1D90-  C9       RET           ；返回调用它的程序，在执行驱动程序中是转
                                入工作子程序[PC=1EA3H]。
```

十二

执行驱动程序有两个出口：1D7CH 和 1D90H，依代号后遇到的字符而异，均转到刚才压入堆栈的工作子程序入口。出口处的状态是：

A　代号后第一个非空格代码 [00H]。

F　表示 A 中代码性质的状态标志：语句结尾符 00H 和 语句分隔符“:”(NC、Z)，数字(C、NZ)，字母或其他符号(NC、NZ)。[Z、NC]

BC　工作子程序的地址 [1EA3H]。

DE　代号在语句中的地址 [79E6H]。

HL　指向语句中代号后第一个非空格代码地址 [79E7H]。

PC　1D1EH [1D1EH]。

堆栈　除执行 GOSUB、FOR……TO语句外，应为空栈。

（四） RUN 工作子程序

本例运行到此，转到1EA3H 的 RUN 子程序。RUN子程序是执行每一个 BASIC 程序几乎都要使用的，但请注意：它是一个命令的工作子程序，而不是执行驱动程序的一部分。

```
1EA3-   CA5D1B    JP Z,1B5D        ；如RUN代号后语句结束（未指定行号）则转入
                                     BASIC指针初始化子程序 [Z,PC=1B5DH]。
(1EA6-  CDC779    CALL 79C7        ；DOS出口。[本例不经过括号内的程序段]
 1EA9-  CD611B    CALL 1B61        ；RUN 行号 先调用BASIC指针初始化子程序。
 1EAC-  011E1D    LD BC,1D1E       ；初始化已清除堆栈，故重新压入返回地址。
 1EAF-  1810      JR 1EC1          ；转入GOTO子程序。(1EC1- PUSH BC……)  )

1B5D-   2AA478    LD HL,(78A4)     ；程序首址取入程序执行指针HL [HL=7AE9H]。
1B60    2B        DEC HL           ；后退一字节。
1B61-   22DF78    LD (78DF),HL     ；保存(程序执行时开始扫描的地址)。
1B64-   061A      LD B,1A          ；为初始化变量类型表设计数器(26个单元)。
1B66-   210179    LD HL,7901       ；指向变量类型表首址。
1B69-   3604      LD (HL),04       ；初始化为单精度变量。
1B6B-   23        INC HL           ；指向下一个单元。
1B6C-   10FB      DJNZ 1B69        ；循环，直到26个单元预置完毕。
1B6E-   AF        XOR A            ；A清零。
1B6F-   32F278    LD (78F2),A      ；清除“捕获错误”标志。
1B72-   6F        LD L,A           ；HL清零。
1B73-   67        LD H,A
```

```
1B74-   22F078    LD (78F0),HL     ；清除逢错转移(ONERR)地址。
1B77-   22F778    LD (78F7),HL     ；清除"出错语句前最后执行过的地址"。
1B7A-   2AB178    LD HL,(78B1)     ；取内存终端指针。
1B7D-   22D678    LD (78D6),HL     ；将字符串区工作指针复位（清除串区）。
1B80-   CD911D    CALL 1D91        ；调用RESTORE子程序(读数指针=程序首址)。
1B83-   2AF978    LD HL,(78F9)     ；取程序结束地址（变量区首址）。
1B86-   22FB78    LD (78FB),HL     ；作为下标变量区首址。
1B89-   22FD78    LD (78FD),HL     ；并作为空闲区首址(清除变量区)。
1B8C-   CDBB79    CALL 79BB        ；DOS出口。
1B8F-   C1        POP BC           ；因为要清除堆栈，先取出返回地址存于BC,
                                    (本例BC=1D1EH, RUN行号调用时BC=1EACH)。
1B90-   2AA078    LD HL,(78A0)     ；取字符串区低端前一字节地址。
1B93-   2B        DEC HL           ；后退二字节。
1B94-   2B        DEC HL
1B95-   22E878    LD (78E8),HL     ；作为语句执行前的堆栈指针。
1B98-   23        INC HL           ；回到字符串区之前。
1B99-   23        INC HL
1B9A-   F9        LD SP,HL         ；将它作为堆栈栈顶地址（清除堆栈）。
1B9B-   21B578    LD HL,78B5       ；HL=字符串库的首址
1B9E-   22B378    LD (78B3),HL     ；存为字符串库的工作指针(清除字符串库)。
1BA1-   CD8B03    CALL 038B        ；置输出设备为显示器。
1BA4-   CD6921    CALL 2169        ；本指令对V2.0无意义。
1BA7-   AF        XOR A            ；A清零。
1BA8-   67        LD H,A           ；HL清零。
1BA9-   6F        LD L,A
1BAA-   32DC78    LD (78DC),A      ；清除进入FOR语句的标志
1BAD-   E5        PUSH HL          ；在栈底存入两个零字节（RUN的标志）。
1BAE-   D5        PUSH BC          ；将保存的返回地址放回堆栈。[(SP=1D1EH)]
1BAF-   2ADF78    LD HL (78FD)     ；取回开始扫描程序的地址指针。[HL=7AE8H]
1BB2-   C9        RET              ；返回(1D1EH或调用的程序)。[PC=1D1EH]
```

由程序可见，RUN 子程序的主要作用并非"执行程序"，而是 BASIC 各指针初始化。当然，执行 RUN 子程序之后，HL已指向程序首址前一字节，并重新进入执行驱动程序，因而开始了对用户 BASIC 程序的解释执行。

（五） 语句链结构的扫描处理

现在才真正开始解释执行 BASIC 程序。首先利用两个语句之间的间隙检测 BREAK 键。因为在程序执行期间监控程序不接受键盘输入，若不安排周期式地检测BREAK键，程序执行中将不能暂停。在语句间进行检测一般能满足需要，但对于某些执行需时过长的语句，BREAK键就会显得很不"灵敏"。

通过测试 HL 所指处是 00H 还是 3AH 以判定当前是在行首还是行中的语句之首，是后者即转入语句体处理阶段。若是行首，则取行指针以检查是否程序结束，取存行号，并可进行跟踪显示。

对程序执行的"跟踪"是 Level II 的原有功能，一经设定跟踪状态，执行每行程序将先显示该行号，以便用户监视程序流程。V 2.0 未使用，初始化将跟踪标志 791BH 置零。用户若将791BH 置为非零，也可以进入跟踪状态。

```
1D1E-  CD5803     CALL 0358        ；在两个语句执行间隙扫描键盘一次。
1D21-  B7         OR A             ；测试输入(按了键则A非零，NZ)。
1D22-  C4A01D     CALL NZ,1DA0     ；如按了键则检测是否是BREAK键。若是则不再
                                    返回，作必要处理后进入等待输入程序。
1D25-  22E678     LD (78E6),HL     ；存本语句之前最后字节的地址。[7AE8H]
1D28-  ED73E878   LD (78E8),SP     ；存进入本语句之前的堆栈指针。
1D2C-  7E         LD A,(HL)        ；取语句前一字节内容，(79E8H)初始化即为0。
1D2D-  FE3A       CP 3A            ；测试它是否语句分隔符（：号）。[NZ]
1D2F-  2829       JR Z,1D5A        ；如是冒号则跳过以下有关指针行号的处理。
1D31-  B7         OR A             ；测试它是不是零字节(00H)。[Z]
1D32-  C29719     JP NZ,1997       ；若此位置亦非零转去显示"语法错误"。
1D35-  23         INC HL           ；指向下一字节[7AE9H]。
1D36-  7E         LD A,(HL)        ；把链指针低字节取入A，以便测试。
1D37-  23         INC HL           ；指向链指针高字节[7AEAH]。
1D38-  B6         OR (HL)          ；测试链指针是否两个零字节(程序结束)。
1D39-  CA7E19     JP Z,197E        ；如程序结束则转去处理后回到BASIC输入程序。
1D3C-  23         INC HL           ；否则指向行号低字节[7AEBH]。
1D3D-  5E         LD E,(HL)        ；低字节取入E。
1D3E-  23         INC HL           ；指向行号高字节[7AECH]。
1D3F-  56         LD D,(HL)        ；高字节取入D；DE=行号值[000AH]。
```

```
1D40-  EB        EX DE,HL        ; HL=行号，DE=程序执行指针值。
1D41-  22A278    LD (78A2),HL    ; 存为"当前行号"。
1D44-  3A1B79    LD A,(791B)     ; 取"运行跟踪"标志。(V2.0未用此功能)。
1D47-  B7        OR A            ; 测试是否非零(非零—跟踪，零—不跟踪)。
1D48-  280F      JR Z,1D59       ; 如非跟踪状态则跳过显示行号的程序段。
1D4A-  D5        PUSH DE         ; 如跟踪先把DE值(执行指针)存栈，将使用DE。
1D4B-  3E3C      LD A,3C         ; 把"<"号的 ASCII码装入A。
1D4D-  CD2A03    CALL 032A       ; 调用显示子程序显示"<"号。
1D50-  CDAF0F    CALL 0FAF       ; 把HL内的行号值转换为 ASCII码并显示。
1D53-  3E3E      LD A,3E         ; 把">"号的 ASCII码装入A。
1D55-  CD2A03    CALL 032A       ; 显示">"号；即在语句执行前显示<行号>。
1D58-  D1        POP DE          ; 取回堆栈中的程序执行指针值。
1D59-  EB        EX DE,HL        ; 恢复程序执行指针 [HL=7AECH]。
```

(六) PRINT和END语句的执行

以下再度进入 1D5AH，取 PRINT 代号(B2H)，往堆栈中压入返回地址 1D1EH，从工作子程序表中查得 PRINT 子程序地址 206FH 并压入堆栈，再取下个字符(跳过空格，取得字母A 的代码 41H，C 标志复位)，借 1D7CH 的 RET NC指令转入 PRINT 子程序。

执行 PRINT 子程序时，在 20B6H 处调用表达式求值程序(CALL 2337H)，得到 A 值 0 (从该子程序返回时 HL 指向 7AF0H 的语句结尾符)，转换为 ASCII 码，送入显示缓冲区 (在紧接着到来的一次中断时，监控程序把它送到显示文本区，由 6847 取去显示在屏幕上)。然后，恢复程序执行指针(HL = 7AF0H)，借 2107 H的 RET 指令又一次回到 1D1EH，开始解释执行第 20 行。

第 20 行的解释执行情况与上述基本相同，不过由于是转入 END 子程序，它会清除堆栈中的执行驱动程序入口地址，结束解释执行阶段，由 1A18H 进入 BASIC 输入程序，等待用户键入新的命令或程序。

如果最末的语句不是 END 语句，那么执行完毕后将再次进入 1D1EH 入口。显然，这一次测试链指针时，将遇到末尾的程序结束标志 0000H，在 1D38H 的OR(HL)指令使 Z 标志置位，于是从 1D39H 转到 197EH，去检测是否从捕获错误状态返回(V2.0未用)，最后仍将进入 END 子程序，结束执行。这也就是 V2.0 系统允许不以 END 作最后语句的原因。

以上是 BASIC 程序一次"执行"的全过程。尽管程序是那样简短，还是需要经历如此繁复的处理，要执行数百条机器指令来解释这十六字节的程序。可见，为了让用户能尽可能简易地"发号施令"，系统是要付出很大的空间和时间开销的。若不是有高速硬件的支持，这种迂回曲折的解释方式将失去实用的价值。

四、错误处理程序

“人机对话”是 BASIC 语言的特色之一。其中最主要的是系统会自动报告程序出错的信息。这种功能是系统的支持程序之一——错误处理程序工作的结果。

在执行驱动程序和工作子程序中，对程序的各种语法元素，参加运算的数据或中间结果，以及系统的各项指针都要进行必须的检查，若发现有错误，便停止解释执行，转入“错误处理程序”。

V2.0 的原型 Level II系统有一套功能很强的错误处理手段，能使 BASIC 程序具备“自行改正错误”的能力。这套手段包括：ON ERROR（逢错转移，或称“错误陷阱”）和 RESUME（继续执行）语句，ERR（错误编号）和 ERL（出错行号）函数，以及一个“错误处理程序”模块。当编程者预计 BASIC 程序某处在运行时可能出错（当然主要不是语法错误）时，便可在这之前设置一个错误陷阱——用 ON ERROR 语句预先指定一个入口行号，并在那儿准备好一个处理错误的 BASIC 子程序。执行中错误一旦产生，系统的错误处理程序便会使 BASIC的解释执行转入用户的这个子程序。子程序可以通过出错行号函数和错误编号函数获知错误的情况，按预先设定的方案分别处理，改正错误，然后用 RESUME 语句返回指定的程序行继续执行主程序。这就使错误能够不露痕迹地得以纠正，也不致中断程序的运行。如果用户不使用这些功能，错误处理程序就中断对 BASIC 程序的解释执行，报告错误性质和行号，回到等待输入状态。

V2.0 虽然没有使用错误陷阱功能，但对它也未作更动，所以错误处理程序中有的部分经常在作无效运行。不过，通过功能扩展手段，很容易将这一潜在的能力利用起来。

进入错误处理程序入口前，寄存器 E 中已由发现错误的程序装入了“错误代码”。程序先在系统工作区存入有关出错语句的各种信息，清除字符串库和出错语句遗留在堆栈中的无用数据，然后根据用户是否设定陷阱，转入不同的处理。

（一） 错误信息表

V2.0系统报告错误性质的字符串，存于 3CE3HH～3E2BH 的错误信息表中。每一项的首字符也经过变码，D7 置 1，以便检索识别。根据寄存器 E 中的错误代码，便可由 3CD4H 的子程序取出显示。错误代码为偶数数列，每一代码可换算出一个“错误编号”。错误编号 = 错误代码 ÷ 2 + 1，范围是 1～23。错误信息表见表 7-3（下划线的 V2.0 未用）：

表 7-3

编　号	代　码	显示信息字符串	意　义
1	00H	NEXT WITHOUT FOR	FOR与NEXT不匹配
2	02H	SYNTAX	语法错误
3	04H	RET'N WITHOUT GOSUB	RETURN与GOSUB不匹配
4	06H	OUT OF DATA	数据已读完
5	08H	FUNCTION CODE	非法调用（功能代码）
6	0AH	OVERFLOW	溢出

续表

7	0CH	OUT OF MEMORY	超出内存容量
8	0EH	UNDEF'D STATEMENT	数组未经定义
9	10H	BAD SUBSCRIPT	下标出界
10	12H	REDIM'D ARRAY	重复定义数组
11	14H	DIVISION BY ZERO	零作除数
12	16H	ILLEGAL DIRECT	非法直接命令(INPUT)
13	18H	TYPE MISMATCH	数据类型不符
14	1AH	OUT OF SPACE	超出字符串区容量
15	1CH	STRING TOO LONG	字符串过长
16	1EH	FORMULA TOO COMPLEX	字符串表达式过于复杂
17	20H	CAN'T CONT	无法继续运行
18	22H	NO RESUME	尚未从捕获错误状态返回
19	24H	RESUME WITHOUT	未捕获错误却遇到RESUME
20	26H	UNPRINTABCE	使用不存在的错误编号
21	28H	MISSING OPERAND	缺操作数
22	2AH	BAD FILE DATA	文本数据类型不符
23	2CH	DISK COMMAND;SYNTAX ERROR	错误使用磁盘命令

V2.0使用的"REDO""LOADING ERROR""VERIFY ERROR"等项错误信息，不经这里的错误处理程序产生，而由工作子程序直接显示，故没有相应的错误代码。

（二） 错误处理程序的反汇编文本

```
19A2-  2AA278    LD HL,(78A2)     ; 取当前行号。
19A5-  22EA78    LD (78EA),HL     ; 存为"出错行号"
19A8-  22EC78    LD (78EC),HL     ; 再存一个副本。
19AB-  01B419    LD BC,19B4       ; 清除堆栈前将继续执行地址存入BC。
19AE-  2AE878    LD HL,(78E8)     ; 取执行出错语句前的堆栈指针。
19B1-  C39A1B    JP 1B9A          ; 转清除文字串库、FOR标志和堆栈中由出错
                                    语句存入的内容，然后按BC值返回19B4H。
19B4-  C1        POP BC           ; 清除1BADH时往堆栈中压入的两个零字节。
19B5-  7B        LD A,E           ; 错误代码存入A。
19B6-  4B        LD C,E           ; 错误代码再存入C。
```

```
19B7-  329A78   LD (789A),A      ; 存入“错误代码”单元。
19BA-  2AE678   LD HL,(78E6)     ; 取前一个语句最后字节的地址。
19BD-  22EE78   LD (78EE),HL     ; 作为“最后执行地址”存贮起来。
19C0-  EB       EX DE,HL         ; 存出错前最后地址于DE。
19C1-  2AEA78   LD HL,(78EA)     ; 取出错行号测试是否直接命令(行号FFFFH)。
19C4-  7C       LD A,H           ; H放入A。
19C5-  A5       AND L            ; 只有A=L=FFH时，A中结果才是FFH。
19C6-  3C       INC A            ; 如A=FFH，则A+1=00H(进位不计)。
19C7-  2807     JR Z,19D0        ; 如是直接命令出错跳过7字节。
19C9-  22F578   LD (78F5),HL     ; 否则存为“最后执行行号”。
19CC-  EB       EX DE,HL         ; HL=最后执行地址。
19CD-  22F778   LD (78F7),HL     ; 存贮“解释执行过的最后字节地址”。
19D0-  2AF078   LD HL,(78F0)     ; 取ON ERROR语句设定的目标语句地址。
19D3-  7C       LD A,H           ; 测试该地址是否为零(非ON ERROR状态)。
19D4-  B5       OR L
19D5-  EB       EX DE,HL         ; 该地址值存于DE。
19D6-  21F278   LD HL,78F2       ; HL=错误捕获标志地址。
19D9-  2808     JR Z,19E3        ; 如ON ERROR地址为零则转去显示错误。
19DB-  A6       AND (HL)         ; 如有ON ERROR地址，须进一步测试错误捕获
                                   标志是否为零(是则A=0)。
19DC-  2005     JR NZ,19E3       ; 如非零则已捕获错误而尚未经RESUME返回，
                                   不能接受另一个错误，转去显示错误信息。
19DE-  35       DEC (HL)         ; 否则把这个标志值变成FFH(捕获到错误)。
19DF-  EB       EX DE,HL         ; HL为程序执行指针(指向待转入语句首址)。
19E0-  C3361D   JP 1D36          ; 转入执行驱动程序,从HL所指地址执行新行。
19E3-  AF       XOR A            ; A清零。
19E4-  77       LD (HL),A        ; 清除捕获错误标志,因为程序将结束执行。
19E5-  59       LD E,C           ; 把错误代码送还 E。
19E6-  CDF920   CALL 20F9        ; 显示换行。
19E9-  21EC3C   LD HL,3CEC       ; HL=错误信息表首址。
19EC-  CDA679   CALL 79A6        ; DOS出口。
19EF-  57       LD D,A           ; D清零。
19F0-  3E3F     LD A,3F          ; A=“?”的 ASCII码。
```

```
19F2-  CD2A03   CALL 032A       ；先显示一个“？”号。
19F5-  CDD43C   CALL 3CD4       ；根据错误代码，显示错误性质字符串。
19F8-  00       NOP             ；六字节未用。
  ⋮
19FE-  211D19   LD HL,191D      ；HL=“ERROR”代码串的地址。
1A01-  E5       PUSH HL         ；存栈。
1A02-  2AEA78   LD HL,(78EA)    ；取出错语句的行号。
1A05-  E3       EX (SP),HL      ；行号换入堆栈，HL=“ERROR”代码串地址。
1A05-  CDA728   CALL 28A7       ；在错误性质字符串后显示“ERROR”。
1A09-  E1       POP HL          ；取回出错语句行号。
1A0A-  11FEFF   LD DE,FFFE      ；令DE=65534。
1A0D-  DF       RST 18          ；比较HL和DE。
1A0E-  CA7406   JP Z,0674       ；如该行号=65534则使系统重新初始化。
1A11-  7C       LD A,H          ；测试行号是否FFFFH(直接命令的错误)。
1A12-  A5       AND L           ；如HL=FFFFH则A=FFH。
1A13-  3C       INC A           ；如A+1=0,Z标志置位,则非程序行中出错。
1A14-  C4A70F   CALL NZ,0FA7    ；程序执行中出错则接着显示“ IN 行号”。
1A17-  3EC1     LD A,C1         ；本指令对错误处理程序无意义。
```

五、常用子程序

V2.0系统拥有大量各种功能的子程序，限于篇幅，本书不可能一一介绍。除在前面章节已经讲过的那些子程序外，这一节再选择一些重要和对读者较有实用价值的，分类进行简介，供读者阅读系统程序和应用开发时参考。有的子程序实际上是多层结构，我们只给出最初入口地址。

（一） 表达式求值程序及各类运算子程序

V2.0的运算功能很强，它支持整型(-32768～+32767)、单精度型(6位有效数字)和双精度型(12位有效数字)的算术和函数运算，逻辑运算，关系运算以及字符串运算。这是大大超出CPU的直接运算能力的。所以，作为“计算”机的系统程序，必须把它们“分解”为最简单的加、减和移位操作的组合，才能由CPU执行。为此，V2.0中准备了一大批适用于各种运算的支持程序。但BASIC的数学表达式是综合性的，具有多元、多层的复杂结构，因此还需要一个用来扫描分析和解释运算表达式的“驱动程序”，这就是“表达式求值程序”。它根据表达式的内容，按照一定的运算规则，组织各种运算程序参加运算。

1． 表达式求值程序。

表达式求值程序主模块地址在2335H～2405H，有多个用途不同的入口和出口。允许以表达式作参数的语句的工作子程序，在程序执行指针HL指向参数首字节时调用本程序。表

达式扫描运算完毕返回，结果存放在工作寄存器 WRA 中，HL 指向表达式之后的单元地址。

BASIC 的数学表达式的键入和显示格式与普通数学算式基本相同，经过代号化，成为由 ASCII 字符、算符代号和函数代号组成的代码串。表达式计算与执行驱动程序在技巧上有相似之处，同样采取连锁循环运行的方式。但程序语句的解释执行是按地址顺序进行的，而表达式计算必须遵从“先乘除后加减”等运算顺序规则。例如表达式：

A + B ↑ (C - D) * SIN(E)

人工计算时，一般都是先分别求出(C - D)和 SIN(E)的值，再依次计算乘幂、乘，最后再和 A 相加。对于以顺序扫描方式解释执行 BASIC 程序的系统来说，这种跳来跳去的运算难度很大。表达式求值程序虽然从左往右地一次性扫描，但不一定按扫描的顺序运算，遇到不能先算的部分，就放到堆栈里存贮起来，待到后面优先计算的部分算完了再取出计算，这就解决了扫描顺序的运算顺序之间的矛盾。

求值程序将整个表达式分割为一个个“组”来处理。以运算符号为界，每个算符连同它左边的运算元素作为一个组。运算元素可以是一个 ASCII 形式的常数，变量，函数或括号里的子表达式（子表达式又按同样的方法分组）。例如上式可分为 4 个组：A +，B ↑，(C - D) *，SIN (E)。表达式求值程序每循环执行一次，只处理一组数据，要循环运行多次一个表达式才能计算完毕。

为了便于决定运算的先后顺序，系统按计算规则按每组的算符换算出一个“算符优先值”。它们是：↑ (7FH)，/(7CH)，*(7CH)，-(79H)，+(79H)，关系运算符(64H)，AND (50H)，OR(46H)。其中，各种关系运算符之间的优先值级别是：< = (06H)，<>(05H)，> = (04H)，<(03H)，= (02H)，>(01H)。逻辑符号NOT不用于两个运算元素之间，故并不作为算符。同 NOT 一样，函数符号、正负号和括号，都是作为运算元素的组成部分先行处理的，因而相当于更高的优先值(函数处理在前，故级别最高)。

这样，两组运算元素之间是否立即进行运算，由它们的优先值比较的结果便可以判定。当后一组的优先级较高时，本组元素就不能与前组计算，需要将它和算符优先值存入堆栈备以后比较和计算用。反之，若本组的优先级较低或与前组相等，本组的运算元素就可同前组进行中间运算。例如A +，B * 二组，既然 B 后有乘号，必然是要先乘以其后的数，故不能与 A 作加运算。而A *，B + 二组，先计算 A * B 是决不会影响以后的运算的。

表达式运算各个阶段的具体步骤如下：

2337H— 进入本程序的最初入口，入口处设定一个虚拟的最低优先值 00H，以备比较。

233AH— 往堆栈中压入当前的优先值，测试内存是否尚有空闲区，然后调用“取运算元素值”子程序(249FH)，将下一个元素值取入(或经运算存入)工作寄存器 WRA。

2346H— 取运算元素后的下一个代号(应为算符)，查表求出其优先值，与从堆栈取出的前一组优先值比较。若大于前一组则往堆栈存入一组运算数据（第一组肯定大于 00H，因此必存），按地址由高到低的存贮内容是：存回上本组优先值，优先值比较程序段入口地址 2346H，本组元素值，数据类型和算符代码，本组的运算程序地址(算术运算 2406H，逻辑运算 25E9H)。运算数据存贮完毕后，恢复程序执行指针 HL 的值，由 23D1H 的 JP 233AH指令重新进入本程序，取下一个运算元素。

从第二组起，若本组优先级不比前组高，则在比较后利用 2384H 的条件返回指令 RET NC将堆栈顶上的前一组运算程序地址弹入程序计数器 PC，进入中间运算。运算子程序将堆栈中的数据取出，和工作寄存器的数据进行运算，中间结果取代原来的元素值成为本组的当前值（本组原算符不变）。运算子程序结束的 RET 指令将原前组堆栈数据最底下的 2346H 弹入 PC，又一次进入优先值比较程序段，再以当前组和堆栈的更前一组相比，重复上述过程。

如此循环往复，到取下个算符时得到的不是算符，而是<D6H 或 >CDH 的任何代码(说明表达式结束)，则按照遇到优先级较低的算符同样处理，利用条件返回指令使前一组的运算子程序地址弹入 PC，将最后元素值同堆栈中前一组进行运算。最后的这个处于"当前算符地址"的非算符，将一次次地迫使堆栈中所有各组逐一弹出，参加运算，直至最后弹出工作子程序调用表达式求值程序的断点地址，返回调用的程序。表达式运算结果已在WRA中。

表达式求值程序通过调用和转入几个次级程序，分别处理不同种类的运算。

2. 取运算元素值子程序(249FH～2531H)。

这个程序对各种可能的运算元素(常数、字符变量名和代号等)一一进行测试，分别送往相应的子程序求值后返回表达式求值程序。如一个元素的首字符不在所测试的合法范围之内，则假定它是左括号(此外的其他字符都是不允许的)，于是取运算元素子程序就反过来调用表达式求值程序对括号内的子表达式求值。这时要从 2335H 入口，以先核实首字节是否真是左括号，如非左括号则在执行 2335H 的 RST 08H 指令时即作"语法错误"处理。子表达式求值返回后，对右括号作语法核实，返回调用本程序的表达式求值程序。可见，在表达式求值程序运行中，可能会多次和多层地"调用自身"，这称为"递归"。

取元素值子程序对小于 D7H 的一些特殊代号先行个别处理。这些代号及其子程序入口如表 7-4 所示。

表 7-4

CDH (正号)-	249FH	2EH (小数点)-	0E6CH	CEH (负号)-	2532H
CBH (NOT)-	25C4H	26H (&)-	7994H	C3H (ERR)-	24D0H
C2H (ERL)-	24D0H	C0H (VARPTR)-	24EBH	C1H (USR)-	27FEH
C5H (*)-	799DH	C8H (MEM)-	27C9H	C7H (*)-	7976H
C6H (POINT)-	0132H	C9H (INKEY$)-	019DH	C4H (STRING$)-	2A2FH
BEH (*)-	7955H				

注：对正号实际是跳过，取下一字符。

代号 D7H～FAH 的数学函数和字符串函数，经过 254EH 的函数求值公共程序，转入各个函数功能子程序。这些子程序的地址表存于 1608H～164FH。各地址如表 7-5 所示。

用户的汇编语言程序，如需对 ASCII 形式的表达式(无函数代号)求函数值，可直接调用 254EH 的函数求值程序。入口条件为 HL 指向表达式前一字节，A 中存有该函数代号。运算完毕后结果在工作寄存器中，HL 指向表达式之后的第一个字符。

如需求某一二进制数值的函数值，可以直接调用有关的函数子程序。调用前先将自变量送入工作寄存器，数据类型代码存入 78AFH 单元。执行后函数值在工作寄存器中。

表 7-5

D7H (SGN)-	098AH	E3H (TAN)-	15A8H	EFH (CINT)-	0A7FH
D8H (INT)-	0B37H	E4H (ATN)-	15BDH	F0H (CSNG)-	0AB1H
D9H (ABS)-	0977H	E5H (PEEK)-	2CAAH	F1H (CDBL)-	0AD8H
DAH (FRE)-	27D4H	E6H (#)-	7952H	F2H (FIX)-	0B26H
DBH (INP)-	2AEFH	E7H (#)-	7958H	F3H (LEN)-	2A08H
DCH (POS)-	27F5H	E8H (#)-	795EH	F4H (STR$)-	2836H
DDH (SQR)-	13E7H	E9H (#)-	7961H	F5H (VAL)-	2AC5H
DEH (RND)-	14C9H	EAH (#)-	7964H	F6H (ASC)-	2A0FH
DFH (LOG)-	0809H	EBH (#)-	7967H	F7H (CHR$)-	2A1FH
E0H (EXP)-	1439H	ECH (#)-	796AH	F8H (LEFT$)-	2A61H
E1H (COS)-	1541H	EDH (#)-	796DH	F9H (RIGHT$)-	2A91H
E2H (SIN)-	1547H	EEH (#)-	7970H	FAH (MID$)-	2A9AH

3． 运算程序。

（1）算术运算公共程序（2406H～248EH）。它根据参算的数据类型，将表达式求值程序的一次中间（算术）运算分配到相应的子程序进行。子程序地址表存于 18ABH～18C8H，根据数据类型和算符代号计算出表地址后取得。它们是：

整数（表首址 18BFH）：加 0BD2H；减 0BC7H；乘 0BF2H；除 2490H；比较 0A39H。

单精度（表首址 18B5H）：加 0716H；减 0713H；乘 0874H；除 08A2H；比较 0A0CH。

双精度（表首址 18ABH）：加 0C77H；减 0C70H；乘 0DA1H；除 0DE5H；比较 0A78H。

用户汇编语言程序也可直接调用这些子程序。对于“（操作数1）算符（操作数2）”的运算，各子程序的入口条件和出口状态是

数据类型	操作数1	操作数2	结果存贮器
整型	DE	HL	加减乘不溢出为 HL，除法和溢出时为 WRA1
单精度	BCDE	WRA1	WRA1
双精度	WRA1	WRA2	WRA1

比较运算的结果都在 A 中：大于（A = －1），等于（A = 0），小于（A = ＋1）。

调用前应将数据类型标志装入 78AFH 单元。如果两个操作数的数据类型不一，应先转换，同精度较高类型者一致。把需转换的数值送入 WRA1，调用 0ADBH（转换为双精度）或 0AB1H（转换为单精度）即可。

（2）逻辑运算（AND 和 OR）子程序：（25E9H～2602H）。

（3）关系运算子程序：逻辑量比较（25B9H～25C3H）；字符串比较（258CH～25B7H）。

（二） 系统功能子程序

V2.0 将自己经常使用的几个子程序入口，放入系统工作区的"零页调用"地址单元，用 RST 指令调用。

1. 核实规定字符(RST 08H)。

(1)程序地址：1C96H。由 0008H 经 7800H转入。

(2)入口条件：HL＝待检测字符所在地址，(PC)＝用作参照目标代码的地址。

(3)程序内容：在 RST 08H 指令(机器码 CFH)后一字节，是检测时用作参照的目标字符码。如 CF29，便是检测 HL 所指的代码是不是 29H(右括号)。如二者相同，则转入 RST 10H 子程序，取入(HL＋1)的字符后，返回调用的程序(跳过检测目标代码)。如二者不同，则经 1997H 装入错误代码(语法错误，E＝02H) 后进入错误处理程序。本子程序是 BASIC 解释程序进行语法检查的重要手段，用来检查语句特定位置的字符是否符合规定。因为这是一种指令和数据混合在一起的特殊用法，反汇编时不要把参照目标代码误认为是下一条指令的操作码，须跳过这个字节再开始指令的反汇编，否则必错。本子程序只使用累加器 A，不影响其余寄存器内容。

(4)出口状态：转入 RST 10 子程序时，HL 不变，(PC)＝原返回地址＋1 (已跳过目标码)。

2. 取下一个字符并检测其属性(RST 10H)。

由 0010H 经 7803H 转入，详见"执行驱动程序"一节。

3. 比较 HL 和 DE 的值(RST 18H)。

(1)程序地址：1C90H。由 0018H 经 7806H 转入。

(2)入口条件：HL和 DE 中都是无符号整数。

(3)程序内容：用 H 与 D、L 与 E 先后进行比较，不产生差值，结果反映在条件标志上。只使用累加器 A，对 HL 和 DE 均无影响。需比较两个正整数时，可分别放入 HL、DE 再 RST 18H。

(4)出口状态：相等，Z；不等，NZ；DE>HL，C；DE<＝HL，NC。

4. 测试工作寄存器(WRA)中运算结果的数据类型(RST 20)。

(1)程序地址：25D9H。由 0020H 经 7809H 转入。

(2)入口条件：78AFH 单元中应有正确的数据类型代码。

(3)程序内容：取工作寄存器数据类型标志单元 78AFH 的内容进行测试，仅使用 A。

(4)出口状态：测试后累加器 A 及条件标志的结果如下：整型－1(NZ，C，N，E)；字符串型 0(Z，C，P，E)；单精度型 1 (NZ，C，P，O)；双精度型 5 (NZ，NC，P，E)。

（三） 数值处理程序

1. ASCII 码行号转换为数值。

(1)程序地址：1E5AH。

(2)入口条件：HL 指向 ASCII 码数字字符串的首字节。

(3)程序内容：本程序用来取 BASIC 程序的行号和语句中的目标行号值。从左向右扫描数码串，逐个把 ASCII 码变为十进制值(－30H)加进用作累加器的 DE，每前进一位 DE 中前面的累加值乘 10，直至遇到非数字字符为止。如数码尚未扫描完毕累加值就超过 6552，则

作“语法错误”处理。这种简易的算法，造成了 V2.0 使用的行号不能>=65530 的规定。

(4)出口状态：数值在 DE 中，HL 指向数码串后第一个非数码(跳过空格)字符。如被处理的不是数码串，或是负数和小数，本程序不作错误处理，但由于 HL 指向负号、小数点和字母等，继续解释执行时检测语句结束符必然发现出错。

2. 任意 ASCII 数码串转换为数值。

(1)程序地址：0E6CH。

(2)入口条件：HL 指向数码串的首字符。

(3)程序内容：本程序处理一切合法的 ASCII 数码串，将它们转换成相应类型的数值。

(4)出口状态：值在工作寄存器 1 中，并置类型标志。HL 指向扫描到的第一个非数码字符。对非法代码的出错处理方法同上。

3. 将值<=255 的 ASCII 数码串或表达式转换为数值。

(1)程序地址：2B1CH。

(2)入口条件：HL 指向数码串或表达式的首字符。

(3)程序内容：本程序用于取入 BASIC 程序中不得大于 255 的参数，如下标、口地址、字符串函数的自变量等。如不是整数或>255，分别作“数据类型不符”、“数据出界”和“非法调用”等错误处理。

(4)出口状态：数值在 E 和 A 中，D=0，HL 指向数码串或表达式后的字符。

4. 将整型数的 ASCII 码串或表达式转换为数值。

(1)程序地址：2B02H。

(2)入口条件：HL 指向数码串或表达式首字符。

(3)程序内容：除值可大于 255 外，其余同上。

(4)出口状态：值在 DE 中，是正数则符号标志 S=0，是负数 S=1。HL 指向数码串或表达式后的地址。

5. 将工作寄存器 1 中的整型数转换为 ASCII 数码串。

(1)程序地址：132FH。

(2)入口条件：数值在工作寄存器 1 中，HL 指向存放 ASCII 码的首地址，B 和 C 均应存入>06H 的值。

(3)程序内容：用该值及每次的余数依次除以 10^4～10^0，每次的商值 +2FH 便是相应位的 ASCII 码，送入目标地址。如目标地址在显示文本区，便可输出显示。B 和 C 的预置是为阻止产生小数点和逗号(本程序调用 1291H 的子程序，每次 B 和 C 均减一，如 B=0 添加小数点，C=0添加逗号，最多可调用 6 次)。

(4)出口状态：HL 指向 ASCII 数码串后面的地址，该字节已清零。入口处的 DE 内容得到恢复。

6. 将工寄存器 1 中的数值转换为 ASCII 码串存入输出缓冲区。

(1)程序地址：0F8DH。

(2)入口条件：数值在工作寄存器 1 中。

(3)程序内容：将数值转换为 ASCII 码存于输出缓冲区(从 7930H 开始)。如需要可再把它们移到别的地址。

（4）出口状态：HL＝数码串首址，DE＝末址，(DE)＝00H。

(四) BASIC 功能支持程序

1. 查找指定行号的程序行。

（1）程序地址：1B2CH。

（2）入口条件：目标行号存于 DE 中。

（3）程序内容：本程序在 BASIC 解释程序中用途很广。查找的方法是沿着程序语句的链指针逐个取行号与目标行号比较，直到某一行号等于或大于目标行号，或者遇到程序结束标志为止。调用时可有两个不同的入口。1B2CH 入口从程序区首址开始查找。1B2FH 入口从 HL 指定地址开始查找，当然，HL 必须是某个程序行的首址（链指针低字节），否则必错。

（4）出口状态：根据标志位 C 和 Z 便可得知查找结果：

C,Z	找到该行（BC＝该行首址，HL＝下行首址）。
NC,Z	目标行号大于所有行号（BC＝HL＝程序结束标志0000H的地址，也就是可供存放新行开始地址）。
NC,NZ	目标行号不存在但有大于它的行号（BC＝行号大于它的第一个程序行首地址，HL＝再下一行的首地址）。

2. 修改行指针。

（1）程序地址：1AFCH。

（2）入口条件：DE＝需要修改行指针第一行首址。

（3）程序内容：本程序用来在 BASIC 程序地址变动后将所有链指针改为当前正确值。方法是在行中（必须跳过行指针和行号）逐字节扫描，寻找行结束符 00H，找到后把它后面一个单元的地址存入前一行的行指针单元，进行到程序结束标志为止。

（4）出口状态：DE＝程序结束标志的地址，HL＝DE＋1。

3. 测试空闲区是否够用。

（1）程序地址：1963H。

（2）入口条件：C 中存有需要的双字节数。

（3）程序内容：将当前空闲区首址加上所需字节数，如超过堆栈指针值，则报告“内存溢出”，否则返回。如将准备使用的空闲区末址存入 HL，可调用 196CH 入口进行测试。

（4）出口状态：空闲区够用时 C 标志置位，入口处的 HL 原样保存。

4. 在堆栈中搜索 FOR 或 GOSUB 的数据组。

（1）程序地址：1936H。

（2）入口条件：在程序语句中调用，搜索 FOR 数据组时，DE 中应有作为搜索目标的循环变量地址，不指定循环变量时令 DE＝0。搜索 GOSUB 数据组时，DE 应存一虚拟的“变量地址”，（如令 D＝FFH），以排除所有的 FOR 数据组。必须保证堆栈中仅有 FOR 和 GOSUB 数据（栈顶为本次调用的返回地址和进入本语句前压入的执行驱动程序返回入口地址 1D1EH，共 4 字节）。

（3）程序内容：跳过栈顶的 4 字节往下搜索堆栈。搜索 FOR 数据组未指定循环变量时，找到第一组为止，否则成组跨过直到循环变量地址相同为止。搜索 GOSUB 数据组时，遇到

FOR 数据组必然跳过（虚拟“变量地址”不可能相同），遇到第一个非 FOR 数据组则应是GOSUB数据组。所以，如用户干预过系统的工作，曾在堆栈中留下“足迹”未清除干净，调用本程序就会出错。

（4）出口状态：搜索 GOSUB 数据组返回时，NZ，A 中应为 GOSUB 代号 91H（不是则有错），HL 指向堆栈中存放 GOSUB 语句行号的地址。搜索 FOR 数据组找到相符的循环变量时，Z，HL指向存放增量符号的地址，DE 保持不变。找不到应有的 FOR 数据组时，NZ。

5. 查找变量。

（1）程序地址：260DH。

（2）入口条件：HL 指向需查找的变量名的首字符，变量名的结束标志与表达式相同。

（3）程序内容：本程序不但能在变量区找出指定名称的变量，无此变量时还能建立此变量，对简单变量和下标变量均适用。只在变量调用不符规定时才作错误处理。

（4）出口状态：变量地址在 DE 中。

6. BASIC各指针重新初始化。

程序地址：1B5DH。

详见“执行驱动程序”一节。

（五） 输入/输出程序

1. 扫描键盘一次。

（1）程序地址：2EF4H。

（2）入口条件：各寄存器内容均应保存。

（3）程序内容：调用 2EFDH 的键盘检测程序一次，然后置标志返回。

（4）出口状态：按键的字符码在 A中，未按键则 A＝0。

2. 检测 BREAK 命令(CTRL- -)。

（1）程序地址：1D9BH。

（2）入口条件：各寄存器内容都应保存。

（3）程序内容：检测键盘一次，未按 BREAK 键，返回，否则转入 STOP 子程序。

（4）出口状态：未按键或按键非 BREAK，NZ，返回调用的程序，按了 BREAK 则中断执行。

3. 显示字符(包括显示控制)。

（1）程序地址：033AH。

（2）入口条件：待显示字符或控制代码存于 A 中。

（3）程序内容：调用 308BH 的显示子程序，将 A 中代码输出显示。各种代码将由315AH的传送程序送到各自的子程序处理。调用本程序前将控制代码装入 A，即可实现对显示的控制。

V2.0 的显示控制代码是：

08H-光标左移一列；09H-光标右移一列；0AH-光标下移一行；1BH-光标上移一行；1CH-光标回到原点(0,0)；1DH-光标回到行首；1FH-清除屏幕；15H-在光标处插入字符；7FH-删除光标处字符。

（4）出口状态：入口处各寄存器内容恢复，光标位置指针 78A6H 中为新的位置值。

4. 清除屏幕。

(1)程序地址:01C9H。

(2)入口条件:无特别要求。

(3)程序内容:调用上一子程序两次,执行控制代码1CH,1FH 的功能。

(4)出口状态:入口处各寄存器内容恢复。

5. 显示字符串

(1)程序地址:28A7H。

(2)入口条件:HL 指向待显示字符串的首字节,字符串以00H 为结尾标志。

(3)程序内容:将字符串送入文字串库,再逐个字符地输出显示。

(4)出口状态:A=0,Z。

6. 以ASCII 数码显示 HL 的值。

(1)程序地址:0FAFH。

(2)入口条件:整数值在 HL 中。

(3)程序内容:把 HL 值送入工作寄存器,转换成 ASCII码,再输出到屏幕。

(4)出口状态:A=0,Z。

7. 蜂鸣器发声

(1)程序地址:3469H。

(2)入口条件:BC 中含有两个输出脉冲之间的延时循环数(决定频率高低),HL 中含有输出脉冲个数(决定发声时间),D 中为调用前输出锁存器副本值。

(3)程序内容:在内循环的控制下,按一定时间间隔向 6800H 的输出锁存器 D0 和D5 写入互为反码的信息(别的位保持原样),使蜂鸣器发声,至外循环结束为止。

(4)出口状态:可忽略。

8. 蜂鸣器短促鸣叫

(1)程序地址:3450H。

(2)入口条件:无特别要求。

(3)程序内容:以固定的 HL=00A0H,BC=0006H,调用发声子程序。

(4)出口状态:可忽略。

9. 打印字符。

(1)程序地址:058DH。

(2)入口条件:打印字符码在 C 中。B 中有用内容须保存。

(3)程序内容:把 C 中代码的字符打印在托架当前位置,如是控制代码则执行相应功能。代码见配套打印机说明书。

(4)出口状态:可忽略。

(六) 语句命令工作子程序的直接调用

BASIC 语句命令的各个工作子程序,用户的机器语言程序中也可调用。方法有:

1. 当 BASIC 程序和机器语言程序并存时,可以调用工作子程序执行一条指定的 BASIC 语句。

例如 BASIC 程序中有 310 PRINT "ASDF"+LEFT$(A$,2) 语句。可以调用

PRINT 工作子程序来执行它。先将执行指针 HL 指向语句中代号 B2H 所在单元地址，在指令RST　10H 后，CALL　206FH，即可在屏幕上显示结果。这种调用方式，因为堆栈中保存的不是执行驱动程序返回地址 1D1EH，而是 CALL 调用的断点地址，所以语句执行完毕时不会连锁执行下个语句，而会返回调用的程序。因为未经执行驱动程序对指针预置，语句出错时就不能正常地进行错误处理，所以应保证语句的正确性。

2. 作为上一种方法的引申，在不存在 BASIC 程序的条件下，可以按 BASIC 语句参数的格式预先存放操作数据，再调用有关工作子程序取入执行。

这种方法能综合 BASIC 和汇编语言二者之长，充分利用系统的资源。例如编制音乐程序，显然以 BASIC 的 SOUND 语句较为便利，汇编语言调用发声子程序是相当复杂的。在汇编语言状态下输出音乐，可先按 SOUND 语句的规则将发声参数以 ASCII 码存贮。如需演奏"5 32 32 | 1 —— |"先在内存某处依次存放以下字符串的 ASCII 码(每个数码和符号各占一字节)：２１，４；１８，２；１６，２；１８，２；１６，２；１４，９（以００H或":"号码结尾)。

将 HL 指向参数字串之前一字节，在　RST　10H 之后　CALL　2BF5H，即可演奏此乐句。

用这种方式，可以方便地调用 CLOAD，CSAVE 和 VERIFY 子程序。文件名前后应有引号，并须以 00H 或 3AH 为结束符。HL 存有第一个引号之前那个单元的地址，RST 10H 后以 CALL 调用。

3. 对于那些不要参数的工作子程序，一般都可以在汇编语言程序中直接调用，以实现其相应的功能。调用前，应执行一条　XOR　A 指令，以清除 A 和 Z 标志。

可用此法调用的工作子程序有：CLS，RUN（只起 BASIC 指针初始化作用），RESTORE，COPY，CLEAR，NEW。此外，NEXT，RETURN 等工作子程序也可如此调用，但须防出错。

4. 一些需要参数的工作子程序，也可以将参数值装入要求的寄存器、系统工作单元或堆栈后，再从子程序的取参数阶段之后的入口调用。

(1)GOTO 子程序。DE＝目标行号值，(SP)＝1D1EH，(78A2H)＝FFFFH（在输入阶段中)。CALL　1EC8H　从指定行号开始执行程序。

(2)COLOR 子程序。分为两个功能使用。A＝图形颜色码，(HL)＝00H，CALL 38A5H　设定图形颜色。A＝背景颜色码，CALL　38C7H　设定背景颜色。

(3)LIST 子程序。BC＝列表首行地址，(SP)＝列表末行地址。CALL　2B32H 列程序清单。

(4)CLEAR 子程序。DE＝字符串区保留字节数。CALL　1E83H 开辟字符串区并使 BASIC 指针初始化。

第八章　硬系统局部改造实验

任何一台计算机都不可能是尽善尽美的。用户经过一段时间的使用，便会感到有若干不尽如人意的地方，低档机的不足之处当然就更多。由于计算机目前还不是一种廉价的消耗型商品，一般用户并不能随时进行“更新换代”。那么，能不能对原来的机器进行扩充改造，使其改善性能，增加功能，从而满足更高的要求呢？这一章主要介绍有关硬系统局部改造方面的一些知识。

一、系统扩充和改造的可能性

(一)　常规扩充

一般说，在设计、制造计算机时，应该为今后可能范围内的扩充和升级留有余地。以LASER 310为例，它在硬件方面备有两个扩充接口，既可用来配接系统外设，也可供用户插接自行开发的各种功能扩展板。例如：用地址译码器、数据锁存器、功率放大器和继电器组成的多路开关量控制板，可用来对各种家用电器进行程序控制；用地址译码器和大容量SRAM组成的内存扩充板，成本将比市售的成品低得多；用地址译码器、数据锁存器和音响集成电路组成的游戏音响发生器，可以大大丰富计算机游戏的音响效果，并且使音响和画面并行输出，简化编程……

在软件方面，系统留有扩充外接软件的空间和软接口。用户将自己开发的系统程序如LOGO、汇编等写入EPROM，与地址译码器等组装成外接软件卡，可以使LASER 310成为多语言系统。此外，还可以通过插卡使之变成单片机开发系统，等等。

以上这些，都属于常规的功能扩充手段，并不需要对系统的主体——主电路板和系统程序作任何变动。本章及下一章要着重讨论的是，在需要和可能时，如何对系统主体进行局部改造，以提高机器的使用性能。

(二)　硬件局部改造的条件

用户自己对计算机硬软件进行改动，并不是常有的事。一方面因为计算机设计严谨周密，经过反复修改和优化，有时简直到了“增一分则长，减一分则短”的境界，可作改动的余地不大；另一方面，计算机的价值可观，为安全计一般也不敢轻易去动它。但任何计算机系统都不是天衣无缝的。就像解牛高手庖丁能够“目无全牛”一样，在深知计算机系统结构的人眼里，它也不过是若干硬软“元件”的组合，到处都有可以下“刀子”的地方。本书作者提倡用户向自己的电脑“开刀”。对于一般计算机爱好者来说，系统局部改造的实践不仅能带来使用方面的实惠，更重要的是作为一种难得的实验机会，通过对“活”的计算机作“手术”，可获得很多生动有益的知识。

当然，我们不提倡对任何计算机都进行任意的改动。改造至少应有以下几个前提：

第一，机器性能有某些显著缺陷而又可能经局部改造得以弥补的。

第二，硬件改造对象以价格较低的低档机为宜，失误的损失不大。

第三，进行前应有一定的计算机知识准备，对目标机的硬软系统有比较清晰的了解。

第四，操作者应具有一定的电子装修操作技能和实践经验。

按前两个条件，本书的标本机 LASER 310 是合适的对象，第三个条件在阅读本书后可基本具备。计算机爱好者多是电子爱好者，如不具备第四个条件，不妨与人合作进行。总之，本书读者以 LASER 310 进行硬件改造的实践，是完全可行的。

（三） 问题分析

从硬件方面将 LASER 310 同中华学习机等相比，它的主要缺点是内存较小和显示分辨率较低，二者又决定了系统不可能具有汉字处理功能。内存扩充是容易以常规手段解决的，插接硬汉卡也不困难，最关键和最难处理的是显示分辨率低的问题。

LASER 310 的屏幕显示最高分辨率为 128×64，用来绘制图形显得相当粗糙，用以显示 16×16 点阵的汉字，全屏只能容纳32个，基本无实用价值。在我国，不具备汉字功能的计算机，用途便受到限制。这可以说是 LASER 310 的一个“致命”弱点。

通过对显示发生器 MC6847 的分析，我们已经知道它具有以更高分辨率显示的潜力，但系统未予使用，通过将 GM0、GM2 接地，GM1接正电源，使图形显示模式固定为 CG2 一种。

但若要提高分辨率，并非只是改变 6847 的接线就能解决的。首先，由于显示是采取内存映像方式，必须有足够存放全屏显示信息数据的显示 RAM 空间。原用的 6116 为 2KB，只够 CG2 模式下存放 128×64 的像素（每像素 2bit）数据。要提高分辨率必须相应增加VRAM 容量。我们可以作一个试验，将 6847 的 GM0 和 GM2 都改接正电源，执行 MODE（1）语句后，即可进入 256×128 的 RG6 显示模式，但屏幕上是完全重复的三条显示图像。这是由于 6847 的访存地址信号 MA12 和MA11对 6116 不起作用，造成它三次重复访问同样的 2KB 范围。虽提高了分辨率，但并不能显示更多的信息。

RG6 模式需要 6KB 的 VRAM 与之匹配。扩充 RAM 的规模不难做到，但它的地址空间将同系统发生冲突。V2.0 的 VRAM 地址之前是基本 I/O 地址，之后是系统工作区，增加的 4KB“无地容身”。若分成两块，显示发生器无法访问，放到内存其余位置，系统程序无法访问，而修改系统程序，工程量就大得多了。

此外，提高分辨率的硬件模块，不可能利用机器的插口以插件形式扩充。因为插口只连通 CPU 的总线，与视频系统无直接联系。除非另做一块视频电路板，工程也太大。

二、 显示改造的基本原理

虽然提高 LASER 310 的显示分辨率有诸多困难，但并不是无隙可乘。几年来，经过计算机工作者和广大爱好者的共同努力，已基本解决这一问题，并先后推出了几个改造方案和扩充系统。其中笔者开发的 PH 高显及汉字系统是其中最为简易的，下面将以它为讨论对象。

（一） 显示 RAM 的扩充

根据 RG6 模式的需要，用于扩充的 VRAM 芯片应考虑以下几点：第一，以静态 RAM 为宜；第二，容量≥6KB；第三，最好是单片，以简化线路。

因此，8KB 的 SRAM 6264 是较为适宜的对象。由于主电路板上已无另外装置 IC 的空位，而且受机箱和屏蔽罩的限制，板子上方和下方也不能堆叠附加的电路板，6264 接入系统

的方式只有两种选择。一是装在机外的电路板上，以带状电缆与机内的视频电路连接，这种方式下6264与6116并存，各为所用，但接线较多是其缺点。另一种方式是直接以6264代换6116。因为6116的24条引脚中，有23条与6264相同位置的引脚兼容，所以代换后只需解决4条引脚的接线问题，比较简单易行，而且可靠性好；但拆卸IC需要一定的操作技巧和经验。我们采用后一方式。

（二） 分辨率选择信号的产生

对6847分辨率选择端的控制，不宜模仿原来采取的"接死"方式，例如把三条引脚都接+5V，虽然简单，但将图形模式固定为RG6，原系统的MODE(1)绘图功能将被破坏。要使分辨率可以任意选择，常规方法是增加一个控制电路。例如，利用一个I/O口地址和三位锁存器，便可以通过向它写入不同的数据而改变显示模式。但对于LASER 310来说，还有更为简便的途径。

从硬件剖析中已知，门阵列电路GA004中含有一个8D触发器，V2.0只用了八个锁存信号中的五个，Q1，Q6，Q7均闲置，系统初始化时置为0，各种工作程序均不改变，因而可以"免费"利用。这三个信号正好控制三个选择端，但带来一个问题，就是它们在初始化都被置为0后，MODE(1)和SET功能将不能执行，破坏了与原系统的兼容性。为此，应不改接GM1，只用Q7、Q6控制GM2和GM0，初始状态便与原系统相同(图8-1)。当然，只控制两位使可选择的图形模式减为四种，但其中已包括原有的CG2，和分辨率最高的CG6、RG6，完全能满足实用的需要。

（三） 显示区地址分配

扩充的显示RAM与6847配合很简单，将高两位的地址线DA12，DA11连接即可。但显示RAM还须由CPU寻址。原显示区地址为7000H～77FFH，代换后就是6264低端2KB存贮空间的地址范围，但其余部分的地址不能向后延伸，因为后面已是系统工作区的地址了。

可见，扩充的VRAM需要使用另外的内存地址。一种安排是放在18KB基本系统的主存贮器地址之后(≥B800H)，为此需要增设一个地址译码器，并且将影响内存的扩充。我们按照尽量挖掘机器内部资源潜力的原则，决定选用基本系统未使用的4000H～67FFH的10 KB空区。这样做有一个极大的好处，因为门阵列电路GA003中的双2-4译码器正好有个4000H～5FFFH的选通输出信号（第13脚，我们将它命名为$\overline{\mathrm{VH}}$）未被使用，这就可以免于增设译码电路。

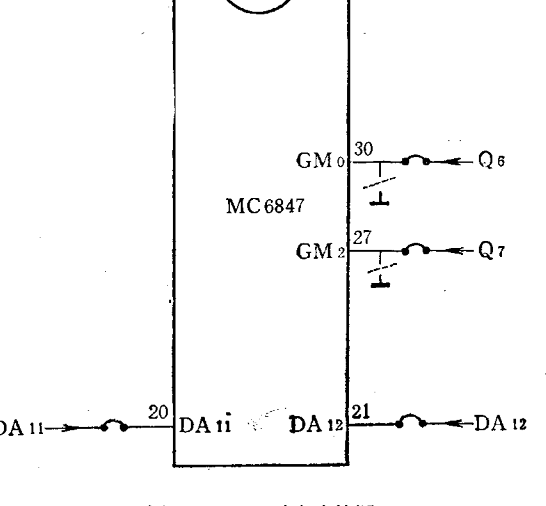


图8-1 6847引脚改接图

以上处理将在两个方面影响原系统的工作。第一，若在硬件上将VRAM片选信号由原来的$\overline{\mathrm{VR}}$（对应地址7000H～77FFH）改为$\overline{\mathrm{VH}}$（对应地址4000H～5FFFH），原系统程序有关显示的指令均无法执行；第二，4000H开始的这片空间是系统保留给DOS的，占用后就再不能加插软盘驱动卡，有的用户利用这片空间

扩充的外接软件也不能运行。

为了解决第一个问题，应使$\overline{VH}$和$\overline{VR}$都能指向6264芯片，解决第二个问题可使DOS同扩充的VRAM切换使用同一地址空间。因为LASER 310执行DOS命令和高分辨率绘图不会同时进行，所以基本无冲突，唯一的缺点是不能在磁盘和显示区之间直接交换信息(需经过中介缓冲区)。

(四) 控制逻辑电路

因寻址范围扩大，6264将连接原来6116不用的A12和A11两条地址线。这时会产生另一个问题：V2.0系统7000H～77FFH的显示区地址信号，A12和A11为"10"。原来它们只是用以参加译码产生$\overline{VR}$信号，对6116不起直接作用。对于6264，最高二位地址为10时则是第三个2KB区段，而6847在文本和CG2模式下访存范围是6264第一个2KB区段。这就使得CPU和6847的信息交换脱节，不能显示。解决的方法是对A12进行控制，当系统使用7000H地址段(即$\overline{VR}$有效)时，将6264的输入地址信号DA12强制置0，逻辑式是$DA12 = A12 \cdot VR$。当$\overline{VR}$有效时，6264的DA12DA11＝00，使显示数据的读写操作都指向起始2KB，与6847访存范围一致。

为解决高显与DOS兼容问题，需要对4000H～5FFFH地址信号加一个"标识"，说明现在的这个地址是指向DOS还是指向VRAM的。这实际上就是用一个切换信号来控制$\overline{VH}$。GA004正好还有一个Q1可供利用。控制逻辑式是：

$VH' = \overline{Q1 \cdot \overline{VH}}$ (表达式中上横线表示反相而不是负逻辑信号，下同)

$VH' = 0$，选通VRAM；$VH' = 1$，选通固化DOS的ROM。为使GA003的八总线收发器由$\overline{VR}$和$\overline{VH'}$均可打开，用或门将二者合并为$\overline{VR'}$，作为VRAM选择信号。逻辑表达式$VR' = VR \cdot VH'$。二者中任一为低电平时$\overline{VR'}$均为低电平，CPU都可对显示RAM进行读写。

对DOS的开关控制，可通过控制送往扩展口的$\overline{MREQ}$信号来实现。逻辑式是：

$MREQ' = \overline{VH' \cdot \overline{MREQ}}$

$VH' = 1$，$MREQ' = MREQ$；$VH' = 0$时，$MREQ' = 1$，外接的ROM或RAM因而均被封锁。这样，DOS和VRAM的信息就不会同时出现在数据总线上了。

以上几个电路综合起来，为节省器件和减少控制板面积，采用一片四与非门IC 74LS00和一片四与门IC 74LS08，组成一块小控制板。其中用一个正与门作为负逻辑或门，两个与非门输入端并接作为反相器；见下页图8-2(*a*逻辑图，*b*电路图)，印板见图8-3。

(五) 磁带机的遥控

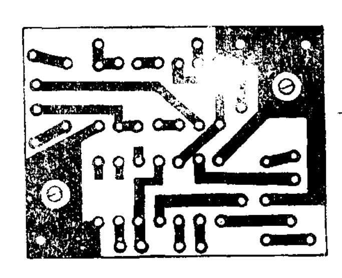
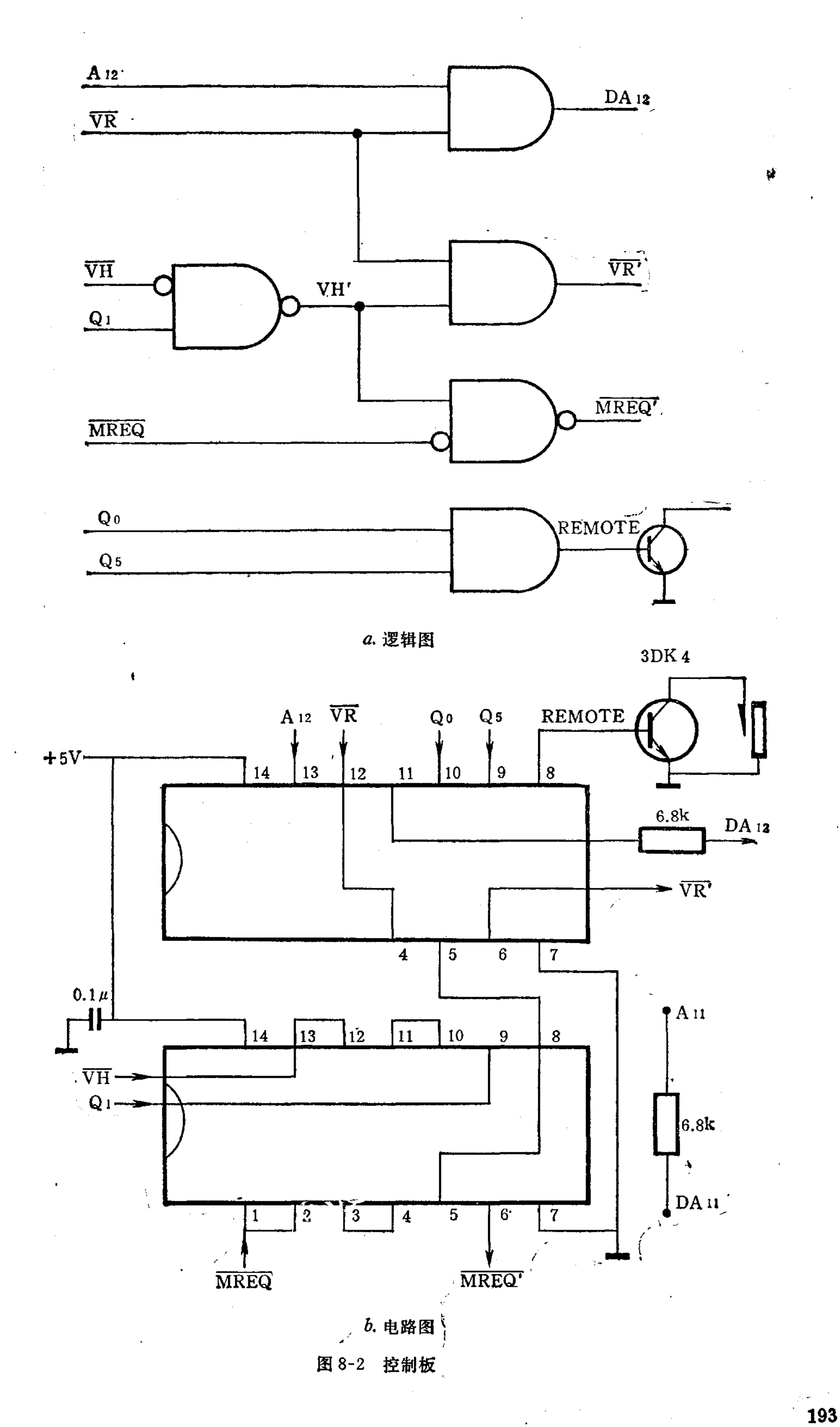


图8-3 控制板印制电路(1:1)

LASER 310配套的磁带机DR 10，同一般计算机用数据磁带机一样具有REMOTE(遥控插孔)，可用来控制磁带的运转或停止。但主机并无此控制电路。为了支持磁带软汉字系统，增添磁带机遥控功能很有必要。

磁带机的遥控插口，实际上是串接在电动机电路里的一个开关。未插入插头时，常闭触点保持电路接通。插入插头后，触点簧片被顶开，电路关断，并可由插头的两条引出线接通(图8-4)。因而若把引出线接到一个开关管的*e*，*c*两端，便可用基极电平控制电



a. 逻辑图

b. 电路图

图 8-2 控制板

路的通断。因为主机的 8D 触发器已无空余输出信号，利用它的发声信号端 Q5和 Q0通过控制板上 74LS08 的一个与门产生 REMOTE 信号，作为开关管的基极输入(图 8-2)。REMOTE 为高电平时，开关管饱和导通，低电平时开关管截止。Q5与Q0从系统初始化开始一直被 V2.0 置为互反电平，因而，REMOTE通常总为无效，不会误开磁带机。增加此电路后，用指令将Q5和 Q0 同置为 1 ，就可启动磁带机，恢复为互反状态则磁带机停。

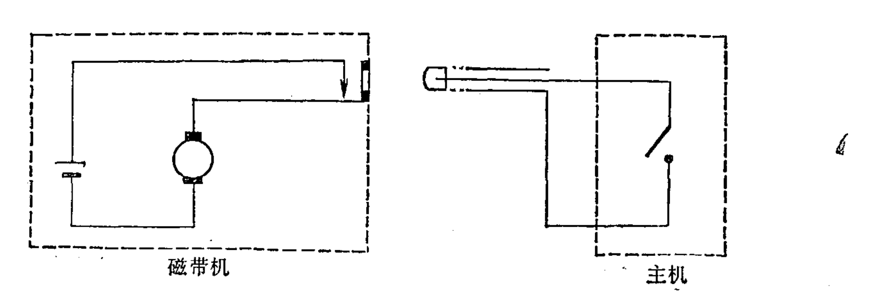


图 8-4　磁带机遥控电路原理

三、 改 装 工 艺

(一)　材料与工具

1. 元器件。

(1)8KB 静态 RAM一片，常见的型号有：日立的HM6264，日电的μPD4364，三菱的M5M5165，韩国的HY6264 等，均可互换。按存取速度分档，选用 150ns(6264-15)即可。CMOS集成电路输入阻抗高，易损坏，拿持时勿接触引脚，存放时用铝箔包好。为避免芯片直接焊接，应配一个 28 脚 IC 插座。

(2)二输入端四与非门集成电路 74LS00，二输入端四与门集成电路 74LS08，各一片。

(3)NPN 中功率硅开关三极管($I_{CM}\geq$300mA，如 3DK4)一只。2.5mm 二芯插口一只。

(4)1/8W6.8K 电阻二只，0.1μF 瓷片电容一只。

(5)40×30mm^2 环氧敷铜板一块，各色软接线共 2m。

2. 工具。

(1)20W～30W 内热电烙铁一把，松香酒精液、焊锡丝若干。

(2)拆卸集成块工具一套。可选用吸锡器、注射针头、专用烙铁附件等。

(3)万用表一块。

(4)手摇钻一具。

(5)废钢锯片及改锥、镊子等。

3. 安全措施。

为确保机件安全，必须采取防除静电、电烙铁漏电及感应交流电的措施。简易方法是埋设可靠地线，保持工具、人体(可戴接地手镯)良好接地，并实行防静电操作。

焊接要快而准，防止虚焊、误焊和反复焊接烫坏芯片。

(二)　操作步骤

1. 主印板改线。

打开机箱，旋下固定印板和电源开关的螺钉，烫开印板同底屏蔽板间的联线。面对机箱后各插口，将主印板向前翻转(图 8-5 附书末，是印板背面(非元件面)的示意图，无关部分略去，以后关于方位的描述均以此为准)。为便于操作，先将蜂鸣器的连线焊下。操作中须注意保存好螺钉，切勿用力扭动键盘输入电缆，以防折断或脱焊造成故障。

印板背面(非元件面)的铜箔线条，有五处先预切断：6847 第 27 脚与地线间；6847 第 27 脚与 30 脚间；6847 第 30 脚与 31 脚间；GA003 第 14 脚与输出线上第一个焊点间；内存扩展插口左起第五脚与板内连线上的第一个焊点间。以上切断点在图 8-5 上用箭头和短线标出。断线工具可用钢锯片，以 45° 角折断，执手端用布包裹。用锯片尖头断面铲断铜箔，用力不要太猛，严防伤及邻近线条。铜箔须铲除到底。断线后用万用表测量其两端阻值，应在数千欧姆以上。完成后将铜屑小心吹拂干净。

印板背面有三条新增连线，可用软接线先行连接。它们是：GA004 第 24 脚连接 6847 第 27脚；GA004 第 23 脚连接 6847 第 30 脚；6847 第 31 脚就近接地。

2. 更换 VRAM。

将印板的屏蔽罩各焊脚一一烫开，去掉屏蔽罩。

(1)用拆卸集成块工具将 6116 拆下，尽量保护焊孔和线条完好。焊孔的残锡要吸净，保持通畅。拆下的 6116 可留作别用。

(2)因为 6264 要比 6116 占更大面积，需将原 6116 前方(管壳缺口方向)的两个电阻、一个 100pF 电容、一个 3.3μH 电感焊下，移到印板背面同一位置，元件脚要穿过原焊孔，与两面的线条都要焊妥，但正面尽可能保持平坦，以便安放 6264 管座。该电感和电容强度较低，易破碎，挟持时需小心。

(3)将印板正面原 6116 芯片底下一个焊点通往它的第 21 脚的线条切断，由这个焊点焊一软线将 $\overline{\text{VRWR}}$ 信号引出以便与 6264 插座第 27 脚连接(见图8-6)。

(4)将 28 脚管座第 1、2、27、28脚弯折向上，除 27 脚连接上述引出线外，第 2 和 28 脚亦先焊上软接线备用。在印板移走电阻的部位垫一绝缘塑料片。然后将管座的 24 只引脚按原 6116 同方向插入焊孔。在焊孔及引脚涂上松香酒精液，烙铁稍多蘸锡，并延长焊接时间，让焊锡充分流布到印板正面，把正面的焊点同时焊好。

(5)将管座第 28 脚引出线焊在缺口前的宽铜箔线条(接 + 5 V)上。将第 2 脚引出线绕到印板背面焊在 6847 的 21 脚上。在印板背面用软线将23 脚(原 6116 第 21 脚)与 6847 第 20 脚相连。

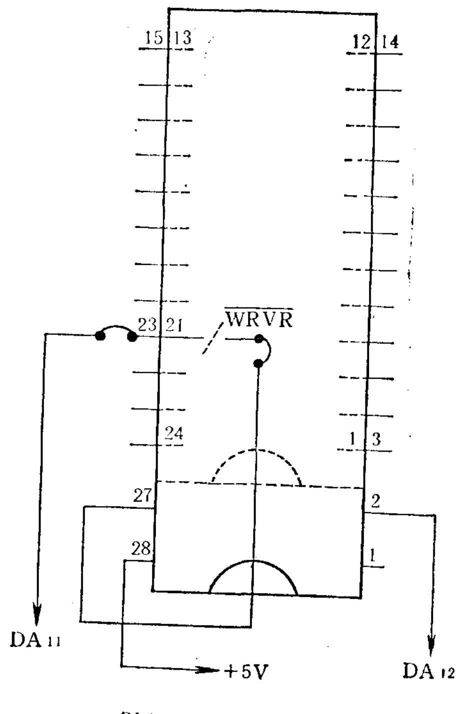


图 8-6 6264代换6116方法

3. 控制板的安装。

（1）按图 8-3 制作印板，插上元件焊好。组装时注意片子和管子各脚不要插反；0.1μF 电容非接地端用飞线跨接 + 5 V 电源。先在控制板上焊齐连线。每条线最好用不同颜色以便区分。线的长度可在图 8-5 上直接量出，取两焊点间直线距离，稍留余量，不宜过长。

（2）与控制板连接的 IC 芯片是：*a*.Z80A；*b*.GA003；*c*.GA004；*d*.MC6847。连接引脚为

1——*d*.20　　2——*a*.19　　*c*.——内存扩充口左 5　　4——地　　5——*d*.21

6——*c*. 34　　7——*c*. 16　　8——*c*. 22　　9——*b*. 14　　10——*b*. 5

11——+5V　　12——*b*. 6　　13——*b*. 13　　14——*c*. 17

如书末附图 8-5 所示，将控制板各条连线的游离端焊到主板相应的位置。图中的小圆圈表示在原有焊点上加焊，黑点表示在铜箔线上增加的焊点。各条线都以始末端为准，图上十字交叉处不连通。

（3）将控制板靠在下机壳内右前角斜面上，正对键盘印板的缺角，试一试闭合机壳应无妨碍，借助于导线的张力即可固定。再在近旁侧壁钻孔固定遥控插口。

4. 磁带机遥控电路。

遥控磁带机的 REMOTE 信号通过 2.5 mm 二芯插孔输出。用两只 2.5 mm 二芯插塞，两条导线分别连通内外芯，磁带机和主机增设的遥控插孔各插一端即可。

如所配的普通录音机无 REMOTE 插孔，可以加装。方法是：切断电动机电源线，两端分别加长引出，焊在二芯插孔上。电源正端接动簧片，负端接外套。

5. 装复和检验。

（1）仔细核对改装电路，如无误即可将 6264 插入管座。盖上屏蔽罩时，6264 前方一电解电容可能翘起过高，应轻轻扭向一侧。屏蔽罩先不焊牢，用二枚螺钉暂时固定印板，以便开机检验。

（2）开机后应一切如常，可执行程序和命令。然后用命令　POKE　26624,234。

屏幕除顶上一条外，应呈类似蜂巢状均匀图案（顶上一条不规则图像在下面应不重复出现）。按 RETURN 键可回到文本状态。

（3）如不能达到上述要求，应检查差错。否则便可装复，改装即告完成。

四、 系列机改装及兼容性处理

LASER　310 系列机型中，305 型印板与 310 型完全相同，可以沿用上述改装方法。早期 200 型社会拥有量较少，改造方法从略（作者愿为用户个别咨询）。

（一）　200（改进）型的高显改装。

该型印板结构与 310 型差别较大，但系统逻辑结构和改造原理基本相同，只是具体操作上有一定差别（见图 8-7，附书末）。其要点如下：

（1）板上线条切断处：6847 第 27 脚与连线间；6847 第 30 脚与 31 脚间；靠近 6847 第 20 和 21 脚中间位置的那个焊点与 GA004 第 20 脚间；GA003 第 14 脚与连线间；内存扩充口左起第 5 脚与板内连线间；原VRAM　6116 第 21 脚与连线间；原VRAM　6116 第 6 脚右上角那个焊点与连线间。

（2）板上改接的连线：GA004 第 24 脚连接 6847 第 27 脚；GA004 第 23 脚连接 6847 第 30 脚；GA004 第 34 脚连接 6847 第 12 脚；原 VRAM 6116 第 21 脚（即 6264 第 23 脚）连接

6847 第 20 脚(换片后再接通)。

(3)换片。因为原 VRAM 前面印板上已无余地,只能用一个 24 脚插座代换 6116。6264 的第 1 、2 、27、28 四脚弯折向上,重叠在前面的集成块上,其余 24 只脚插入管座。用软线将第 28 脚焊接在与 26 脚相通的宽线条(接 + 5 V)上;第 27 脚用一短线焊通它正下方的焊点(连通 GA004 第 25 脚);第 2 脚连线绕到印板背面焊到 6847 第 21 脚。

(4)控制板连线逻辑位置与 310 型相同。

2. 200型内存的扩充。

LASER 200 型的主存贮器只是一片 6116(或 2016),除去系统工作区的几百字节,用户区不到 2KB。但系统设计时在印板上保留了两个 24 孔的扩充插座,插入两片 6116 (与基本 RAM 方向相同)焊好即可使用,系统分配的地址是 8000H～87FFH 和 8800 H～8 FFFH。在 ROM 旁边也留有一个 28 孔的插座,是准备用户扩充 ROM 的。可利用来简便地再扩充一片 6116。具体方法是:第一,先将其第 23 孔与 ROM 第 23 脚(A11)间的连线切断;第二,将 6116 插入后部 24 孔(不用 1 、2、27、28 孔)焊好;第三,用软线将这片 6116 的第 18 脚接 GA003 第 12 脚(9000H片选信号),第 20 脚接Z80A 第 21 脚($\overline{RD}$),第 21 脚接 Z80A 第 22 脚($\overline{WR}$)。于是它就成为 9000H～97FFH 的读写存贮器。实际接线如书末附图 8-7 所示。

3. 高显改装与系统兼容性。

LASER 310 改装为高分辨率显示后,与原来的硬软件及外部设备是完全兼容的,均可以照常运行。其中,只有两点需要特别予以讨论。

高显系统与 DOS 的兼容,需要软件的支持。二者的切换操作应在机器语言级进行,而不能使用 BASIC 的 POKE 语句写 6800H 的手段。因为系统接有 DOS 时,系统初始化就将"取下一字符"子程序入口改到 DOS 内部的 4293 H,以便识别非代号的 DOS 命令。若已将输出锁存器的 Q1 置 1 进入高显状态,DOS 即已关闭。这时发出的任何 BASIC 命令,在执行驱动程序调用 RST 10H子程序时,都会误入 VRAM 中与 4293H 等效的地址,造成迷路死机。用机器指令进行切换才能避免这个问题。

LASER 310 经上述改造后,会出现同系统配置的 64K 内存扩充卡不兼容的现象。这是因为采取控制 $\overline{MREQ}$ 向外部输出的方式来控制 DOS 的工作。64K 卡也需要这个 $\overline{MREQ}$ 信号作为选通条件。插有内存扩充卡时,系统初始化已将堆栈设置在内存高端的扩充 RAM 内。当 Q1 置 1 时,外接存贮器被封锁,堆栈丢失,系统工作遭到破坏,造成锁死或复位。

因此,当配用 64K 卡时,应将 $\overline{MREQ}$ 输出线上的断点焊通,供 64K 卡使用。控制板的 $\overline{MREQ}'$ 信号,由内存扩充口左 5 脚改接到左起第 11 脚的空位上,用来控制 DOS。

软盘驱动卡需要相应作一点改动:打开卡盒面盖,插口向上,在印板非元件面上部的竖立排线左端下方,有一组 16 脚的 IC 焊点(74LS138)。将其中左排上数第 4 个焊点的输入线条铲断,将此焊点同插口处第二排左起第 11 个焊点连通。这样,控制板的 $\overline{MREQ'}$ 信号,便可送入外接 ROM 译码器的控制端。



第九章　软系统功能扩充的手段

比起硬件来，对系统软件功能的扩充要容易得多。事实上，很多电脑用家都不仅限于应用软件开发，常常研制出一些扩充系统程序功能的辅助性软件，使自己的计算机使用起来更加得心应手。功能扩充可在不同的层次上进行。最高级别的扩充系统是与基本系统平行的、完全独立工作的第二(或更多个)系统程序。其次是在基本系统的基础上扩充的高级模块，例如DOSV1.2之于V2.0。再次是功能子模块的扩充，例如为BASIC解释程序增添一些新的语句命令工作子程序或函数求值程序。最后，还可以对系统一些子程序的内容进行修改和补充。不同层次的功能扩充手段，也可能重叠和交叉使用。

本书的标本机LASER 310，几乎在推广应用的同时，对它的系统程序V2.0"二次开发"就展开了，并在80年代末期形成热潮。原因主要有以下几方面：

第一，V2.0的BASIC解释程序有较明显的局限。它虽在Level II的基础上增添了一些功能，但同时又"抛弃"了前者不少有用的功能。系统程序使用内存多达16KB，而功能较之12KB的APPLE SOFT和PC-1500的BASIC系统却有所逊色。用户深感不足，希望有所扩充。

第二，V2.0系统的"可扩展性"比较高。一般的计算机软件系统都是可以扩展的，基本系统的子程序资源为扩展系统准备了良好的条件。而V2.0对扩展来说更是得天独厚。首先，它保留了Level II扩展磁盘系统的手段而未用，用户可以方便地使用它们；其次，系统中"埋藏"着不少未被使用的工作子程序，发掘出来就能轻而易举地增加许多功能。此外，系统可用于功能扩展的软接口也比较多。

第三，随着硬系统显示分辨率改造的实现，在软件方面增加高分辨率绘图和汉字功能势在必行。没有软件支持，高分辨率就失去使用价值。

1985年以来，国内不少计算机科技工作者投入了对V2.0系统的二次开发，并取得不少成果。作者也曾先后开发出P1.5A，P1.5B和PH等版本的扩充系统软件，在全国推广应用。本书将结合作者的实践，进行深入具体的讨论。

一、　充分发掘系统的潜在资源

由于计算机研制中的种种具体情况，一个系统的可使用功能未必就把它的内部能力发挥殆尽。系统功能扩充首先应着眼于内部潜力的发掘，因为这是最为经济和有效的。在这方面，LASER 310可以说是个特别突出的例子，大有潜力可挖。

(一)　系统指针和参数的利用

系统工作区里的各个数据，是系统运行的基本依据，BASIC用户一般毋须过问，更不可任意加以改变，否则可能造成灾难性的后果。但在我们已经确切地了解它们之后，系统工作区就不再是"禁区"了。有意识地利用里面的信息，甚至改变一些系统指针和系统变量的值，就可以

轻而易举地获得某些新的功能。下面是V2.0的几个实例。

1. 程序运行跟踪。

从剖析执行驱动程序得知,解释执行一个新行时,由1D44H开始,要访问作为跟踪标志的791BH单元。如该单元内容非零,则调用0FAFH的子程序,先把当前行号显示出来,再处理语句本身。由于V2.0系统初始化时已将跟踪标志清零,又没有提供跟踪命令,所以此功能形同虚设。可见,只要我们在791BH单元内装入任何非零字节,立即便可实现程序跟踪功能。例如:

```
POKE  31003,1
RUN
```

随着BASIC程序的运行,将显示执行到的每一行号,如<10> <20>……,以便用户监视程序流程。

跟踪标志不会被NEW、RUN和CLEAR清零。退出跟踪状态的方法是 POKE 31003,0。

2. 自动产生行号。

BASIC输入程序从1A39H起,是自动产生行号的程序段。查看"自动设行"标志78E1H单元若为非零,则从78E2/78E3H取出当前应产生的行号值,显示于行首。再从78E4/78E5H取行增量值与之相加,如未超过65529,存入78E2/78E3H供下行使用。然后,接受用户键入的语句内容……

所以,只要将起始行号装入78E2/78E3H,行增量值装入78E4/78E5H,再使78E1H为非零,即可进入自动产生行号的模式,直至行号超过65529或按下BREAK为止。

使用这一方法时,置自动设行标志值的步骤必须放在最后,否则你将没有设定初始行号和增量的机会。BASIC指针初始化子程序和BREAK命令都能清除自动设行标志,但不改变其余四个字节的内容,如重置79E1H单元为非零,则将按中断的行号系列继续往下进行。

3. 变量类型定义。

V2.0规定,不加类型说明符%、$的变量一律视为单精度类型。它只支持双精度常数运算,而不允许用#号定义双精度变量,在需要较高精度的场合,很不方便。系统剖析已说明,系统建立一个变量时,如它不带类型说明符,就根据变量名第一个字母,到系统工作区7901H～791AH的变量类型表中取得类型代码,并据以开辟变量值的存贮空间。变量类型代码可为2、3、4、8四种。但BASIC指针初始化时将整个表都预置为4(单精度)。

所以,只要把变量类型表的某一字节置为别的类型代码,就能使所有以相对应的字母开头的变量都成为所指定的类型。因为RUN命令会使变量类型表重新初始化,故此法最好采取程序语句的形式,放在程序之首。例如,要将以A、B、N、M四字母开头的变量分别定义为整型、字符串型、单精度型和双精度型,可用以下两条语句:

```
10  DATA A,2,B,3,N,4,M,8
20  FOR I=0 TO 3:READ X$,X:POKE 30912+ASC(X$),X:NEXT
```

此后,这些变量不带任何标志都是指定类型,带类型说明符者除外。

4. 恢复读数据指针到指定行。

V2.0的RESTORE语句使读数据指针复位到程序起始地址前一字节。当程序很长、

DATA 语句较多、数据类型混杂时，从 DATA 语句中读取所需数据比较困难。如能够刚好把数据指针恢复到所需数据之前，是再方便不过的了。

系统的读数据指针在 78FF/7900H 二单元，RESTORE 子程序的作用不过是把程序之前一个字节的地址装入其中，此后的 READ 语句即从这里开始寻找和扫描 DATA 语句。所以我们也可以模拟此法，将一个 DATA 语句的首址 - 1 存入其中，系统便会由此开始读取数据。

在 BASIC 程序中使用这种手段效率虽然低一些，但有时可以简化编程。例如，用 PP 40 绘图仪打印向量型汉字，将 N 个字的笔划向量数据依次分别存放在行号为 0～255 的 DATA 语句中。然后有下面的程序段：

```
300  INPUT N:DIM X%(N-1):J=-1
310  FOR I=31464 TO 32767:IF J=N-1 THEN 340
320  IF PEEK(I)=0 THEN J=J+1:X%(J)=I:I=I+5
330  NEXT
340  ······
```

这样就把每个字的地址放在一个数组中，下标是各自的行号，也就是它的打印代码。在打印单个汉字子程序的开头，应为：

```
1000  Y=INT(X%(N)/256)
1010  POKE 30976,Y:POKE 30975,X%(N)-Y*256
1020  〔汉字打印子程序〕······
```

要打印哪一个 DATA 语句的字，LET N=〔打印代码(即行号)〕:GOSUB 1000 就可打印出来。

5．BASIC 错误处理。

V2.0 的错误处理程序，只使用了显示错误信息、返回输入阶段的部分，未给用户提供 BASIC 级的错误处理手段。这一点可以通过直接干预系统工作区来加以弥补。

78F0/78F1H 二单元可用来存放一个 BASIC 语句出错时的转移地址。V2.0 在 BASIC 指针初始化时将它们清零，因而失去作用。如向其中存入作错误处理的 BASIC 子程序地址，则程序出错时将不再中断运行和显示错误，而是转入这个错误处理子程序去继续运行，以识别和处理错误。错误的性质代码，在 789AH 单元，出错的行号，在 78EA/78EBH 单元中。

能够转入错误处理子程序的另一个必要条件，是“错误捕获”标志 78F2H 单元为零，否则虽有转移地址也不接受错误处理。

错误处理子程序结束后，可用 GOTO 语句转入继续执行地址。如需让错误处理子程序继续有效地工作，转移前应将 789AH 清零。

6．查程序行的地址。

扩充 BASIC 功能，有时需要了解程序行的地址。一般 BASIC 版本均无“行地址”函数。通常可以用一个 BASIC 子程序，沿着链指针进行搜索测试。这里介绍一种更简易的方法。

系统工作区的 78ED/78EEH 二单元，在 BASIC 程序出错时用来存放出错语句前一字节的地址。如果我们需要了解的程序行正好有错，那么执行到这一句并显示错误信息后，

78ED/78EEH 的内容加 1，便是它的首地址。所以，我们只要人为地给语句制造一个错误，就可以很容易地查出它的地址了。

例如，有以下程序行：

420 PRINT A

用屏幕编辑方法在行号后面加一个非法的字符，如 * 。变成

420 * PRINT A

用 GOTO 420 执行后，显示"SYNTAX ERROR IN 420"

再用命令串

X = PEEK(30959) * 256 + PEEK(30958) + 1:PRINT X:POKE X + 4,32

即显示该行地址并用空格覆盖掉那个非法字符。注意如 X 可能>32767 时，应增加"负地址"转换的内容。

（二） "废弃"代号的起用

V2.0 的代号范围为 80H～FAH，从代号表可见其中有 49 个代号未被使用。查看内存的保留词表，这些未用代号的相应位置有数量不等的零字节，可见它们的保留词是被故意"抹掉"的。然而，这种处理并不彻底。在工作子程序地址表中，这些代号的工作子程序入口地址仍旧保留着。这就是说，系统的代号化子程序"不认识"这些保留词，但执行驱动程序却"认识"这些代号。因此，只要"绕开"代号化过程，它们的功能还是能够使用的。

参照 Level II 系统，这些可用代号原来的保留词，大部分已用下划线字符列于第 7 章的"工作子程序地址表"和"函数子程序地址表"中。它们的功能和用法，请参阅下一章。其中有两项需特别说明：

第一，SYSTEM 的意义与 Level II 不同，其"工作子程序"已被改为以 0000H 为入口，实际上等于系统重新启动。

第二，代号 9EH 可对应于两个不同的保留词，各有不同的工作子程序。当它出现在语句之首时，执行驱动程序按查表所得地址转入 SOUND 子程序执行。如它跟在 ON 代号之后，则被 ON 子程序作为 ERROR 代号，转入 ONERR 子程序执行。这是 V2.0 对 Level II的修改造成的。

另外，还有一部分"磁盘 BASIC 代号"，留待下一节讨论。

如果将上述"保留词"写在程序语句中，代号化子程序因为检索失败，就把它当作变量名原样保存下来。所以一经执行，便会出错（函数有的则被作为变量对待）。但若代号化以后的语句中出现这些代号，执行驱动程序却完全能够正确执行。所以，利用这些代号功能的方法，便是直接将代号置入 BASIC 程序区的程序语句中。为此，键入程序行时可先用一个字符占据未来代号的位置，语句送入程序区以后再找到它的地址，用 POKE 命令置换进代号。占位字符一般宜选择不常用到的，如"&"、"\"等，以便识别，图案符号通不过输入程序，故不能用。另外，代号在语句中的位置必须符合语法。

例如，我们需要调用 V2.0 未用的 MEM 函数子程序，取当前内存空闲字节数，可用程序

10 PRINT&

这个 & 号地址为 7AEEH，用 MEM 代号 C8H 代换它：

POKE 31470,200:GOTO 10

立即执行后即显示空闲内存字节数。但 LIST 时,该行变成了:

10 PRINT

原因是代号 C8H 没有相应的保留词,所以代号及以后的内容都不能显示,但并不影响程序的执行。屏幕编辑时切不要在这一行按回车键，重新输入就等于把未显示的内容删掉了。

那么，在直接命令中能否使用这种办法呢？回答是肯定的。因为立即执行语句可以由以"："号分隔的几个语句组成。执行前面的语句给后面语句替换代号就可以了。其关键是代换地址要算准。命令串的形式是:

POKE 代号地址,代号:扩充功能语句

直接语句是存放在输入缓冲区，代号化以后立即执行。代号化语句的首址是79E6H，POKE 等代号只占一个字节。如要使用 AUTO 功能,可用

POKE 31217,183:&10,10

显示器下一行立即自动产生第一个行号(光标在后):

10 ■

用这种方法,不需要专门的软件,就能在 V2.0 的基础上增添 21 种语句命令和函数功能,纯属"废物利用",简便易行。不过,当扩充功能语句处于程序中间时，确定替换代号的位置比较麻烦。可采用前面介绍过的一些手段来解决。例如,含有占位字符(如&)的语句,一执行就会出错,很容易由错误参数找到行地址,也可以用一个错误处理子程序扫描出错行，自动替换代号。甚至可以在语句中直接使用扩充的保留词，执行中由错误处理子程序到预置的扩充保留词表里检索,替换代号,多余的字母位置用空格覆盖。还可以进一步用 RESUMR 0(代号化)语句,回到已经放好扩充代号的语句去重新执行。这样，扩充的保留词就可以合法地在程序中使用,造成的错误被"掩饰"过去,不会中断程序的运行。

至于置入代号语句会"消隐",难于记忆和修改的问题，也有一个变通的办法,那就是先制造一个"副本"备案。在用字符占位的程序行键入回车后,把光标调回这一行,将行号加 1，行号后增加一个"'"号 (与"REM" 等效)，再回车。于是原行后面就有了一行语句体相同的副本。然后对原行进行置入代号操作。因为副本行是 REM 语句,不被执行，自然也不会出错,任何时候都能列表显示。修改已经消隐的语句时,只需修改副本行,回车后光标调回，把行号减 1,删去"'"号,再回车,便有了新的一行程序及其副本了。

(三) 空闲存贮单元的利用

系统对存贮器进行分配时,总是考虑一切可能的用途。但实际运行中往往只使用一部分。暂不使用的存贮单元就可以供扩充系统临时借用。V2.0 的系统工作区是按 Level II 划定的,有的单元实际并未使用,我们完全可以利用起来。

1. 显示文本区。

LASER 200～310 的显示文本区都为 2K 字节,在低分辨率显示模式下,只使用低端的512字节。如果程序不需要进入高分辨率模式，这闲置的 1536 个字节便可用于存放机器语言程序或数据。这样做同 BASIC 无冲突,但不允许使用 MODE(1)——它将清除显示区的全部内容。

2. 输入缓冲区后部。

Level II 划定的输入缓冲区是根据它的最大行长256字节，V2.0将高端派了别的用场，还剩181字节。接受从显示区取入的当前行，最多只需64字节。LIST将测试行复原成ASCII格式时可能有超出64个字符的情况。后部大部分空间是不会用到的，可资利用。

3. 系统参数区。

除个别未用单元外，7847H～787CH一段空间中，仅784CH用作磁带操作提示信息开关，其余均可供用户使用。

4. 磁盘BASIC出口区和DOS出口区存贮单元的利用见本章第三节。

二、 功能扩充程序

扩充系统功能的最基本方法，就是编制一些机器语言程序，对系统程序的功能进行补充、辅助以至修正、取代。但以支持BASIC为基本任务的系统，一般不具备支持机器语言程序输入、汇编、存贮、编辑和运行的手段。这是需要首先解决的。

（一） 开发工具

在机器没有配置处理机器语言的系统程序时，只能依靠BASIC的有限手段。用户先将机器语言程序手工汇编，并将机器码换算成十进制数，写成DATA语句或以INPUT语句由键盘输入，再用POKE语句装入指定内存单元。POKE语句中的十进制地址参数，必须在整型数的范围之内，即相当于16位有符号二进制数。0000H～7FFFH相当于0～+32767，8000H～FFFFH则相当于-32768～-1。所以64KB内存的0～65535个地址中，大于32767的地址都要转换为“负地址”才能用于POKE语句和PEEK函数，方法是：负地址=正地址-65536。这样，用户程序区的32767地址单元之后的地址便是-32768，-32767，-32766……，仍然呈升序排列。在循环中装入时必须注意在上述节点进行转换。

采用上述方式是比较费力的，容易出错而且难于纠正。

近几年来，已先后有几种LASER 310的开发工具软件问世。较早的是青岛刘晓明开发的“MONITOR”系统程序。它具有“小汇编”功能，可将用户逐条键入的Z80符号指令汇编为机器指令（不支持标号和伪指令），此外还有反汇编、数据装入、数据块移动、设置断点等多种功能。它对后来的LASER 310二次开发起了一定促进作用。

1990年，河南张涛将PC机的DEBUG程序移植到LASER 310上。DEBUG是一个融编辑、汇编、调试等功能为一体的集成软件，共有19条命令，包括汇编、反汇编、编辑、读写内存、数据串查找、数据块比较和移动、程序运行（含单步运行）、设置断点、显示和修改寄存器内容、十六进制运算和数制转换、打印输出、磁带读写校对等等。

它除能进行小汇编外，还能汇编带行号的源程序，因而用户可以像BASIC一样地编写、输入和编辑修改汇编语言程序，可以使用标号和Z80汇编伪指令（除DEFL），系统经两次扫描源程序，生成机器语言目标程序。它具有友好的用户界面，便于使用。例如，将光标移到任一显示行回车，即可输入该行的命令；可在屏幕跟踪汇编过程，源程序有错会显示出错行号，待用户修改后继续汇编；系统保留了七条单字符命令和相应的软接口供用户扩充功能。整个系统程序仅占3.1KB内存，对用户程序影响不大。建议读者采用。

（二） 用户保留区的开辟

由于V2.0不支持机器语言程序，并未给它划分存贮区域。任意将它放入当前的空闲区是不行的，因为那样很不安全，易遭BASIC破坏。解决这个矛盾，可采用强行改变BASIC某些划分内存的指针值，使一部分内存空间脱离系统的控制，成为用户保留区，用来存放机器语言程序。

1. 后移BASIC程序区起始地址。

BASIC程序起始地址指针78A4/78A5H是在系统初始化时被设定为7AE9H的，系统解释执行BASIC程序都以它为依据，没有任何子程序会审查、改变其内容。所以如果人为地把它改变为更高的地址，系统亦会同样地予以承认，对于7AE9H至现在首址间的内存空间，将不再问津。例如：

POKE 30884,77:POKE 30885,123

此后，程序区便以123×256+77=31565(7B4DH)为首址。7AE9H～7B4CH这100字节空间的控制权为你所有。

后移程序区首址，有几个问题必须注意：

第一，如果程序区已有BASIC程序并需保留，不能简单地用此法改变首址指针，因为那样会破坏BASIC程序，使其无法运行。要开辟保留区一般宜在输入BASIC程序之前进行。

第二，改变程序区首址指针值后，必须把新的程序区之前的那个单元和开始的两个单元内容都清为零，才能正确地装入和运行BASIC程序。如上例，可用

POKE 31564,0:NEW

第三，CSAVE命令只能将BASIC程序区的内容写入磁带，对保留区无作用。在程序首址后移的情况下记带，必须记住此时的程序首址。调用这个磁带文件之前，须同样对系统作改变指针处理。否则，程序装入地址与当前首址指针不同，将不能列表和运行。对任意起始地址的磁带文件，运行前把装入首址指针(781E/781FH)的值移进程序首址指针(78A4/78E5H)即可。这样处理以后，有时LIST无异常，而RUN却不能运行，显示“语法错误”。原因是程序首址前一字节不为零(详见“执行驱动程序”)，将其清零就一切正常了。

一般说，用户保留区里总是存放着供BASIC程序调用的机器语言子程序或数据，需要同BASIC程序一起记带并在今后一起装入内存。这样最好采取以下步骤(仍以保留100字节为例)：

第一步，在程序区首址为7AE9H的时候，先键入一行“磁带文件启动程序”：

0 POKE 30884,77:POKE 30885,123:RUN

第二步，改变程序区及其指针(如前述)。

第三步，键入BASIC程序，往保留区装入机器码。注意保留区前面30字节已被启动程序占用，用户只能使用31495以后的空间。

第四步，记录磁带前将程序首址复位，用命令串：

POKE 30884,233:POKE 30885,122:CSAVE“〔文件名〕”

这样记录的磁带文件便可在正常的程序区装入，由预置的0行程序将指针改变到位，然后启动主程序运行。

2. 后移变量区起始地址。

变量区首址在BASIC程序结束标志0000H之后那个字节，其值存在78F9/78FAH中。系

统存放变量以这个指针为依据，决不会超越。所以，如果把这一指针值在实际的基础上加大，就可以在BASIC程序同变量之间辟出一片保留区，存放机器语言程序和数据。这种方法的最大好处，是保留区的内容可以同BASIC程序一起用正常方式记带，以后正常调入运行。但使用时应注意以下问题：

第一，与前一方法相反，一般须在程序完全定型以后进行，如已执行过，要先CLEAR清除变量。保留区装好数据后如再增删程序，其首址和内容会遭到破坏。这就使得保留区装数据比较困难。只好用POKE作直接命令装入。也可以在这时后移程序区，另外运行一个装入程序，装完再恢复原址。

第二，改变变量区指针，除简单变量首址外，还包括下标变量首址和终址指针，对后者只须执行一条CLEAR命令即可。如用此法开辟1KB空间的保留区：

POKE 30970,PEEK(30970)+4:CLEAR

3. 前移内存终端地址。

由系统剖析得知，RAM的物理终端地址是系统初始化时实际测试而得，指针在78B1/78B2H，V2.0系统将字符串区建立在它之上，始终不会改变它的值。当机器接有软盘控制卡时，DOS的初始化过程会将此指针值减少311字节，使内存高端成为DOS工作区，得到保护。

仿效这一办法，即使已有DOS，仍可再向前移。此法的优点是保留区与BASIC各自独立存在，互不相扰。缺点是记带和以后调入都需当作两个程序分别进行。改变内存终端指针值后，字符串区和堆栈指针均应随之改变。保留1KB空间的方法是：

POKE 30897,PEEK(30897)+4:CLEAR 字符串区保留字节数

注意字符串区的字节数决不可省略，因为不带参数的CLEAR不改变字符串区首址指针值，那样的话，堆栈将保持在原来地址，会破坏保留区的内容。

三、与基本系统的接口

系统的功能扩充，除需要适当的扩充程序之外，还必须解决它同基本系统的关系问题，使二者能按要求切换运行和相互调用。

V2.0系统程序的固化文本独占了机器启动的入口，尔后又一直在封闭式的系统中无休止地循环往复运行。要让扩充程序获得工作机会，必须采取措施“打进”主系统的循环中，窃取对机器的控制权，与主系统“轮流坐庄”式地交替工作。

扩充程序不应该同基本系统完全隔绝。对系统缺乏深入了解的用户，虽然也可以开发出一些与之毫无联系的独立的Z80机器语言程序，需要时用USR函数调用，在硬系统的直接支持下运行，但这无疑是一种低水准的扩充方式。因为基本系统是一种可贵的软件资源，有大批功能子程序可供使用。例如，扩充程序的数据处理不知利用系统中现成的运算子程序，而是另起炉灶自设一套，便会造成内存空间的浪费。尽可能地利用基本系统的软件资源，是功能扩充的一项基本原则。

从扩充程序调用基本系统的子程序是很容易的，但对于扩充的系统功能子程序或原有子程序的扩充部分，则是反过来应由基本系统调用它们。这就需要利用系统中的软接口，把扩充程序“嫁接”在基本系统的主干或分枝上，在主系统对软接口作例行访问时得以工作。计算机系统软件一般都备有各种软接口，供系统扩充之用。V2.0的软接口是比较丰富的，除系统有

意识设置的以外，有些是系统设计中无意留下的“可乘之隙”。这种情况，给V2.0的二次开发提供了优越的条件。

（一） 预留扩充软件入口

V2.0为软件扩充预留了4000H开始的10KB内存空间，并设有三个软件扩充接口。接口地址分别为4000H，6000H和8000H(8000H接口只对200型适用)。接口只宜用来联结固化软件。由接口地址开始的四个字节内容必须是接口标志——AAH，55H，E7H，18H。基本系统在初始化最末阶段逐一查询接口标志，发现某一接口已接上扩充软件便转入第五字节开始执行。

这个接口占有比BASIC优先的地位。LASER 310扩充的磁盘操作系统DOS V1.2便以它和V2.0联结。DOS V1.2的内容主要是DOS命令检索程序、工作子程序和软磁盘驱动器控制程序。进入4004H后，实际上只是执行了DOS的初始化程序，开辟出DOS工作区，设置DOS命令测试入口(见下)，然后回到V2.0的BASIC输入程序。以后基本控制权仍在BASIC，只是在检测发现DOS命令时才转入DOS，DOS命令执行完毕又回到BASIC中。用这个接口扩充其他系统软件也可仿此进行。

（二） 中断的使用

中断，是现代计算机技术中常用的系统模块切换手段。在中断开放的条件下，它可以有力地插入主系统的循环之中。每一个中断信号对应一个中断服务程序入口，每个入口可以连接一个系统功能模块，利用中断便能转入执行。按中断的优先级形成的中断链，把各个模块按轻重缓急有机地组织起来。

V2.0的监控程序，便是利用中断进行系统切换的实例。LASER 310仅有一级可屏蔽中断能力，监控占据后别的扩充系统就不能利用了。不过V2.0在中断服务程序入口处为用户留有一个软接口(CALL 787DH)，可用来连接别的扩充系统。这个接口是在视频的$\overline{FS}$有效后调用，所以最宜用于扩充屏幕显示功能，以避免输出干扰屏幕显示。

当系统扩充了高分辨率绘图和汉字显示功能时，有关语句产生的显示数据应先存放在缓冲区内，而把高显数据输出子程序的入口向量置入787DH。每次进入中断，将首先执行高显数据输出子程序，将显示码送入VRAM，然后返回执行监控程序的其余部分。

（三） 字符&的使用

在V2.0系统中，字符&是很特别的一个。它的ASCII码26H是唯一小于80H的代号。不过它本来是用于Level II的磁盘BASIC的，在V2.0中并无用处。它可以跟在任何允许用表达式作参数的语句代号之后，由表达式求值程序转到7994H的一组出口。将一个机器语言程序入口地址存入7995/7996H，便可通过&而转入其中。

需要注意的是，&不能用作直接命令或放在BASIC语句之首，否则将被解释程序作为“语法错误”。因此它比较适宜用于语句参数的功能扩充，如扩充的函数，十六进制参数等。若需借以扩充工作程序，一般用PRINT & 的形式调用。这里的 PRINT 是不得已的“多此一举”。进入 PRINT 工作程序到转入&入口之前，堆栈里压入了调用表达式求值程序的断点、虚拟的优先值和调用取元素值子程序的断点等三组16位数据。进入扩充程序后应该先把它们清除掉，程序执行结束才能返回执行驱动程序的1D1EH入口。

（四） “磁盘BASIC出口”的利用

V2.0的原型Level II安排给磁盘BASIC的28个代号，本系统全部未用，但从工作子程

序地址表中仍可查得它们的"地址"。这些地址集中于7952H～79A5H，每组三字节，可供存放一条指向其工作子程序入口地址的JP指令。V2.0系统初始化时，已将它们都置为JP 012DH（显示"磁盘命令错误"子程序入口）。如果我们把一组地址换成一个机器语言子程序的入口地址，当BASIC执行到相应代号时，便可由此转去执行它。这样就能再给系统增加28种新的语句命令和函数。这些代号中小于BCH的语句命令代号有15个，其余都是函数代号，不能用于语句之首，否则为语法错误（见表7-2和表7-5中"#"号部分）。

例如，用代号替换法虽已挖掘出系统的绝大部分潜力，但仍有几个工作子程序派不上用场，原因是V2.0增添的保留词把它们的代号占用了，工作子程序地址表中原属它们的单元，现已换成新的住户。不过我们既有28个备用代号的"编制"，当然也可以把这些"隐士"请出来工作。这些闲置的工作子程序及其地址是：

1DF7H- TRON 跟踪

1DF8H- TROFF 关闭跟踪

1FF4H- ERROR 模拟错误

1E00H- DEFSTR 定义字符串变量

例如在PH扩充系统中，依次使用代号A7H～AAH，预置出口地址为：

```
7988-   C3F71D   JP  1DF7
798B-   C3F81D   JP  IDF8
798E-   C3F41F   JP  IFF4
7991-   C3001E   JP  1E00
```

通过这些代号，便可调用有关功能。不拟使用的出口向量单元，可以用来存放数据或短小的程序。因为正常情况下程序中不会含有相关的代号，也不会转到这些地址来。

（五） "DOS出口"的利用

前述的功能扩充方法，仅限于发掘和增加BASIC功能，还不便用以修改原有的工作子程序的功能。其实，V2.0的某些子程序是具有可修改性的。

在系统程序的某些地方，有一种看来似乎没有意义的调用。例如在BASIC指针初始化子程序中1B8CH是一条CALL 79BBH指令，而79BBH又是一条RET指令，使调用一无所获。这种调用，本来是用作Level II与TRSDOS沟通的出口，79BBH～79BDH在DOS初始化时存入一条JP指令，指向DOS的有关子程序，于是BASIC的初始化便能增加DOS所需内容。LASER机的DOS V1.2并不用这些出口，系统初始化时已全部将它们的首字节置为返回指令码C9H。因为DOS出口在子程序中，所以，我们便有了"钻进子程序肚子里去"的手段。利用得当，能巧妙地修改系统的某些功能。不过，DOS出口毕竟不是为LASER 310的二次开发而设，它们数量有限，分布不均，使得利用的范围也受到限制。

DOS出口的转移向量集中于79A6H～79E4H的一片内存空间，每组三字节。见表9-1。

只要在一组单元中存入一条指向你的机器语言子程序入口地址的JP指令，便能给出口所在的子程序增加内容。当然，必须小心地保护断点处各寄存器的值。返回指令使执行回到DOS出口的下一条指令。

例如，V2.0的USR函数是调用机器语言程序的主要手段。它的自变量是传送给机器语

表 9-1

DOS出口	调用的程序	CALL指令地址
79A6H	错误处理	19ECH
79A9H	USR	27FEH
79ACH	BASIC输入	1A1CH
79AFH	未用	无
79B2H	BASIC 输入	1AA1H
79B5H	BASIC输入	1AECH
79B8H	BASIC输入	1AF2H
79BBH	BASIC初始化，END	1B8CH，1DB0H
79BEH	PRINT(TAB)	2174H
79C1H	字符输出	032CH
79C4H	扫描键盘一次	0358H
79C7H	RUN行号	1EA6H
79CAH	PRINT@	206FH
79CDH	PRINT	20C6H
79D0H	PRINT	2103H
79D3H	PRINT	2108H，2141H
79D6H	INPUT	219EH
79DCH	READ	222DH
79DFH	READ，LIST	2278H，2B44H
79E2H	无	无

言子程序的数据(存到WRA 1)中，并不是入口地址。当不传送数据时自变量无意义，可取任意值。而入口地址则必须先用 POKE 语句装入788E/788FH二单元。需调用的子程序若不只一个，操作是相当麻烦的。如能改为以自变量指定调用入口，自然方便得多。分析系统程序得知，USR子程序的入口处(27FEH)就是一个DOS出口(CALL 79A9H)，TRS DOS就用它改变了USR函数的功能。我们可仿此设置一个扩充程序段，取自变量值存入788E/788FH，然后转入原程序的2805H继续执行。

将扩充程序段入口向量置入79AA/79ABH，并把79A9H内容改为JP指令操作码C3H。以后就能用"USR(子程序入口地址)"的形式调用。扩充程序段如下：

```
RST     10H          ;取下个字符，指向左括号。
RST     10H          ;跳过左括号，指向入口地址首字符。
CALL    1E5AH        ;将地址值取入DE。
```

```
RST     08H                 ;核实右括号。
(29H)                       ;右括号目标代码。
LD(788EH),DE                ;地址值装入用户程序入口指针。
JP 2805H                    ;转入USR子程序。
```

DOS出口还能用来作为扩充BASIC语句命令的通用接口。用户都知道，V2.0 是不接受保留词范围之外的非法“命令”的。输入程序对这种“命令”不能代号化，将原样保留在程序中。执行驱动程序发现后把它作为变量名，转入LET子程序。LET子程序在字母后核实等号不符，就报告“语法错误”。因为执行驱动程序和LET子程序中都没有DOS出口，只能从BASIC 输入程序和错误处理程序中想办法。

BASIC 输入程序1AA1H的 DOS 出口处于程序行已经代号化、尚未送入用户程序区之间。在这里接上扩充程序，重新扫描代码串，仿照基本系统的方法将扩充保留词变为合法代号，压缩程序行，然后返回断点传送程序行。为了解决行的“消隐“问题，还要利用LIST子程序中2B44H的DOS出口，，增添将代号还原成扩充保留词的程序段。

V2.0错误处理子程序中19ECH的DOS出口，处于显示错误信息和返回等待输入阶段之前。将扩充系统的入口接于此处，凡被V2.0认为非法的语句都会转入扩充系统，进行鉴别和处理。若是功能扩充语句造成的，则清除“出错”留下的各种信息，执行扩充的工作子程序后返回执行驱动程序继续往下解释执行，若不是功能扩充所造成，则返回错误处理程序，按常规错误处理。

不使用或不能使用的DOS出口向量，后两单元可用来存放扩充系统的参数。因为前面有RET指令的屏蔽，数据是很安全的。

（六） RST 10H指令的利用

V2.0将Z80的RST 10H指令定义为“取下一字符”子程序，并在系统中大量使用。在执行驱动程序的1D5AH入口处，就是用这条指令去取语句体的首字符。因为RST 10H 的中转地址在系统工作区，是可以重新设定的，所以它成为V2.0中用来扩充语句命令的最为有效的软接口。LASER 310配置的DOS V1.0/1.2，开发的“MONITOR”、“PH1.0-1.3”都是用它与V2.0连接。在扩充系统初始化时，就用一个扩充程序的入口取代7804/7805H 中原来存放的1D78H。于是每次执行RST 10H都进入扩充系统。

系统大量使用RST 10H指令，但只有1D5AH处的这条指令是应该进入扩充系统的。所以扩充系统入口处首先应予鉴别。方法是从堆栈取调用处的程序断点测试，若不是1D5BH，则属一般调用，应即转入1D78H执行取下一字符的操作。只在发现确系1D5AH调用，才进一步检索扩充的保留词并进行相应处理。用这种方法，扩充的语句命令可以不必代号化，就能交付解释执行。当然，因此执行每一次RST 10H都要增加一些检测操作，会减慢系统的工作速度。不过实践证明，在人的主观感觉上差别不大。

下面是 DOS 命令检测程序的反汇编文本。DOS 初始化时将它的入口地址 4293H 置入7804/7805H单元。RST 10H 指令每次都要增加以下内容：

```
4293-  D9        EXX            ；保存各寄存器内容。
4294-  215B1D    LD HL,1D5B     ；令HL＝V2.0执行驱动程序入口地址。
4297-  D1        POP DE         ；取出调用本程序的断点地址。
4298-  B7        OR A           ；将Cy清零以便作减法运算。
4299-  ED52      SBC HL,DE      ；以减法测试断点是不是执行驱动程序入口。
429B-  D5        PUSH DE        ；先存回调用断点地址。
429C-  D9        EXX            ；恢复各寄存器内容。
429D-  C2781D    JP NZ,1D78     ；若断点不是1D5BH则转入取下一字符常规操作。
42A0-  E5        PUSH HL        ；是执行驱动程序调用，保存扫描指针值。
42A1-  CD781D    CALL 1D78.     ；取语句体首字符。
42A4-  2002      JR NZ,42A8     ；若不是00H或：号则转入DOS命令检索程序。
42A6-  D1        POP DE         ；遇到语句结束符则清除堆栈中的HL指针值。
42A7-  C9        RET            ；返回1D5BH,等于执行了“取下一字符”子程序。
```

用这个接口还可扩充多个系统。例如PH系统为了同DOS兼容，在它的初始化时就将7804/7805H的DOS入口地址先存贮起来，再置入自己的入口。每次进入先检索PH系统的扩充保留词，若不是再取回DOS入口转入执行。这样，便可以同时使用V2.0、DOS和PH三套语句和命令了。

同时，这一接口还能用来扩充函数功能。因为取运算元素值子程序入口的249FH也有一条RST 10H指令。在扩充系统中增加测试这一断点的程序段，证实是取元素值子程序的调用、便转入扩充函数子程序处理。

第十章　PH扩充系统程序详解

1989年前后，国内先后推出了几个LASER　310汉字系统，如北京张保田的“QZ”系统，四川罗章寿的“LC”系统，贵州彭辛岷的“PH”系统，以及北京庄志荣的“万能模块”等，都各具一定特色，拥有相当数量的用户。

本章将对PH系统进行透彻的剖析，通过这一实例帮助初学者了解系统程序的设计和功能扩充的技巧。

一、扩充的功能

（一）系统概貌

1．PH的特色。

与别的几种汉字扩充系统相比，PH系统有以下特色：

（1）费用最省。它是目前唯一的仅需投资二三十元便可实现高分辨率绘图及汉字显示的实用系统，真正达到了“让每一位用户、每一台机器都用得起”的目标。

（2）兼容性好。与200～310全系列主机、全部配套外部设备、16K/64K扩充卡、V2.0系统程序、DOSV1.0/1.2磁盘操作系统，以及原有的各种应用软件完全兼容。

（3）环境条件要求最低。不需加插硬卡，不要求配置软盘驱动器和扩充内存（指310型），在主机＋电视机＋录音机的环境下即可实现系统全部功能。

（4）可扩展性强。未占用I/O口和干扰用户区，便于扩充RAM及I/O设备。软件为积木式结构，可根据不同需要装卸各种功能模块。以磁带为软件载体，软系统扩充或更新的费用甚微。

（5）具有国内唯一的磁带软汉字库，调字方法新颖直观，无需借助别的编码体系。

（6）具有遥控磁带机走停的功能和多达12条的磁带文件操作命令，从而在一定程度上克服了内存空间狭窄而磁带机作外存又不便使用的缺陷，恰当地组织磁带文件，便可在小内存中自动运行任意规模的大软件。

（7）高分辨率绘图（含造型表）语句与APPLE　II相同，便于移植其软件；除因硬件所限，X≤255、无HCOLOR语句外，较其增加了在正反相背景上画、抹点、线，移笔和造型显示速度控制等功能，绘图也快得多。

（8）其主要缺点是：由于内存有限和磁带运转需时较多，汉字使用的便利性明显地低于别的硬汉卡系统，因此不宜用于汉字文书处理（对于占绝大多数的、未配打印机的用户，这个问题本来也不存在），但实践证明，在屏幕显示中用作汉字菜单、提示、说明等，完全能够胜任。

2．系统磁带。

LASER　200～310高分辨率显示及磁带软汉字系统（简称PH），由以下部分组成：主机电路改造；功能扩展控制板；磁带软件。

PH系统程序先后推出1.0,1.1,1.2和1.3四种版本,均向下兼容,用法相同。对于不同的机型和内存配置,备有适配型号,如A型(310基本配置),B型(200＋6 KRAM),C型(310＋64K卡)等。以下主要讨论PH1.3A。

PH1.3系统程序的各个模块存于磁带A面。下面是以磁带计数器读数为序的文件目录(各种录音机和磁带读数不尽相同,仅供参考)。

002 "PH1.3"——磁带软汉字操作系统及扩充BASIC解释程序(含有5个子文件PH-1～PH-5,自动联锁调入),018备用副本。

034 "DRAW"——造型表绘图功能扩展模块(附:038"DRAW-DEMO"——演示程序)。

041 "RENUMBER"——重编行号功能支持程序。

062 "CREATE/EDIT"——建立字库与字库编辑程序(附:046"C/E REM"——汉字说明显示程序,052"REM-ZK"——说明文本的汉字库)。

068 编程常用汉字库"ZK0"～"ZK3"。

用信号线和遥控线将磁带机同主机连接起来,放入系统磁带,按下PLAY或LOAD键,应不走带。这时还不能使用扩充的磁带操作命令,只能用6800H的软开关控制磁带机,由键盘发出命令POKE 26624,33: CRUN。磁带运转,装入主文件"PH1.3",然后由它自动装入五个子文件PH-1～PH-5。装载完毕磁带机停,屏幕显示版本标志,进入等待输入状态。此后便可使用扩充的BASIC语句命令。

本系统保留LASER 310原操作系统V2.0以及DOS V1.2的全部功能,基本语法规则相同。系统使用V2.0的全部工作区,用户不能随意介入。

(二)高分辨率显示及绘图

1. 高显基本操作。

(1)颜色设定:PH的高分辨率显示模式是6847的RG6,有两套颜色可供选择。用COLOR,0设定为浅绿/深绿;用COLOR,1设定为白/黑色。每套颜色中以谁为背景色,由清屏命令设定。

(2)高显清屏命令。

① HGR 正相清屏: 进入高显状态; 将绘图原点复位(0,0); 将汉字显示光标复位(0行0列);清除全屏幕;将屏幕基相定义为深绿(COLOR,0)或黑色(COLOR,1)。

② HGR# 反相清屏: 将屏幕基相定义为浅绿或白色,其余同上。

③ 自动部分清屏:从PH的高显模式回到V2.0的MODE(0)或MODE(1)模式时,V2.0系统将自动把它们使用的显示区(7000H～71FFH或7000H～77FFH)内容清除,其余部分的高显画面数据不受影响。当再次从中、低分辨率模式进入高显,PH系统也自动清除被使用的显示区,其余部分的画面仍可重现。

(3)高显画面的保持:高分辨率绘图和汉字显示语句执行后,图形和汉字画面能否保持,有三种情况:第一,它们单独作直接命令,或作为命令串的最后一条命令时,高显画面可自动保持,按BREAK方回到低显文本状态;第二,虽作直接命令,但后面还跟有别的语句,画面不保持;如需保持,可在命令串的末尾再加上一条HPLOT语句,使之变成第一种的情况;第三,作程序语句使用,将随程序执行结束返回低显等待输入状态,如需保持可用程序延时或无限循

环的方法。执行程序中的低显语句亦会退出高显状态。

(4)高分辨率绘图命令。

HPLOT 其后不跟参数,作用是进入、保持或重现高显画面。

HPLOT X,Y 在指定坐标画一点(使该点与屏幕当前背景反相)。X为列数,取0～255(比APPLE II机少24点,移植其程序时需作调整),Y为行数,取0～191,均可为常数、变量或表达式,下同。

HPLOT# X,Y 抹去指定坐标的点(使其与屏幕背景相同)。

HPLOT X,Y TO X,Y TO X,Y…… 在指定坐标点间连续画线。

HPLOT TO X,Y…… 在前次所画终点(或屏幕原点)和指定坐标点间连线。

HPLOT X,Y TO# X,Y…… 抹去指定坐标点间的线段,也可在连续画线中"移笔"。画斜线时,因舍入误差,逆向画线或抹线不一定与正向所画重合,所以一般应从画线的同方向抹线。

(5)高分辨率像元检测:用HPOINT(X,Y)函数。如该点是浅绿或白点,函数值为1,如是深绿或黑点,函数值为0。

造型表绘图功能见本章第(六)节。

(三) 汉字功能

按系统设计的前提,为了既不用硬汉卡又不配软盘驱动器,只能采取磁带软字库。为了减少磁带机和磁带字库操作上的不便,系统已采取了一些弥补措施,还需要用户在实践中逐步熟悉,掌握使用技巧。

1. 磁带软汉字库。

PH1.3磁带B面为基本字库型的国标一级软汉字库(GB5199.1-85,16-55区,3755字,每个文件2区,188字)。字库排序按以下规则:首先按汉语拼音字母(A～Z);同音按声调(阴平、阳平、上声、去声);同调按首笔(横、竖、撇、点、折);同首笔按笔划多少。下面给出的磁带机计数位置仅供参考。

001 16-17啊～炳;014 18-19病～楚;028 20-21础～叠;042 22-23丁～服;

056 24-25浮～哈;071 26-27骸～箕;086 28-29肌～浸;101 30-31尽～傀;

117 32-33馈～窿;133 34-35隆～摸;150 36-37摹～吡;167 38-39啤～渠;

185 40-41取～绳;203 42-43省～塔;222 44-45獭～威;241 46-47巍～晓;

262 48-49小～瑶;283 50-51摇～誉;306 52-53浴～政;331 54-55帧～座。

基本字库型文件专供建库/编辑(C/E)程序调用,作为建立用户字库的数据源。用户程序不能直接使用。

为了减少用户建立字库的工作量,磁带A面备有常用字库"ZK 0"～"ZK 3",收纳编程常用汉字740个。其中"ZK 0"为用户字库型(W)文件,调入内存即可使用,其余为基本字库型文件,用途同B面。

2. 用户字库的建立。

普通磁带机对汉字库磁带不能随机搜索,而小内存中又不可能存放全部汉字数据。因此PH系统采用程序字库方式支持汉字显示。在一个BASIC程序编制完成并存入磁带后,按程序所需的全部汉字,调用建库程序建立一个相应的用户字库,存于磁带程序文件之后。这样,

程序调入运行时，便可将字库装入内存配合工作。程序字库容量有限，因而 PH 系统的汉字功能主要适用于软件中的菜单、提示、说明、注解等，不宜用于大量汉字的文书处理。

一位用户编程所需汉字，一般不过在几百字范围内。所以每位用户都可以先建立起自己的磁带常用字库，以后在它们的基础上建立程序字库，就不必每次都到浩瀚的基本字库中从头搜索，可以节省很多功夫。

建立用户字库的步骤：

（1）列字库表：建库前要先列出所需汉字清单。程序中使用的汉字词语、句子，尽量按词句结构排列，以便作词组快速调用，如内存允许，有的字可以重复。各字按顺序编出字号。每个用户字库不能超出 256 字，最大字号 255，此表供编程时查字号用。

（2）列查字表：将字库表中的字按上述国标字库排序法重新排列一遍。各字仍标出在字库表中的字号。这样可保证搜索字库一次完成，不致因漏字而多费周折。如准备一张区位码汉字表，则可在所需汉字上一一标出字号，搜索时更能一目了然。

（3）调入系统磁带上的 CREATE/EDIT 程序并启动运行，利用其功能搜索磁带软字库，取所需汉字装入用户字库区。一般可先从磁带 A 面的常用字库取字，其中没有的再有限地调用基本字库文件，可节约不少时间。

（4）用户字库按计划装入全部汉字后，按 BREAK 键退出 C/E 程序，用 SAVEW 命令将用户字库记入磁带保存，备今后与程序配合使用。

3. 汉字的使用。

（1）程序字库的调用。程序中用 HPRINT 显示汉字时，必须有程序字库支持。程序字库可在程序输入前、后或运行中由磁带调入。如果一个程序字库字数不够使用，可在程序运行中调入第二个或多个程序字库，全部或部分覆盖以前的字库。所以用字多少并无限制。为此字库的设计必须同程序设计紧密配合。

因程序字库规模小，每个汉字就用它在字库里的排列顺序，也就是字号作为调用的代码。若字库里部分汉字按词组排列，用“首字号＼末字号”的形式可调用整个词组。

（2）高显“光标”：汉字及高分辨率字符显示，有一个类似文本模式中“光标”作用但不显示的指针（78B0H/78B1H），随时指出屏上的下一显示位置。它与低显光标、HPLOT 指针并行不悖地各自发挥作用。在高显状态下屏幕分为 12 行，每行 32 列，每个 ASCII 符占一列，每个汉字占两列。行末不够显示一个汉字时自动跳到下行。HGR 命令可使高显“光标”复位。显示过屏幕末行末列，屏幕内容自动上卷一行，首行消失，末行清空，光标移至末行之首。

（3）HPRINT 语句。本语句可进入高显状态，显示汉字及高分辨率 ASCII 字符（包括原系统不具备的英文小写字符）。保留词后跟显示项目表，语法规则与 V2.0 的 PRINT 相同。如 HPRINT 语句后是“:”号或行结束，则显示行的剩余部分全部清为空格，将高显光标移至下行行首。如语句以“;”“,”号结尾，则行末不清空，分号时高显光标移至已显示的最后一个字符后面，逗号时光标移至下一显示段（屏上每显示行分为两段，第 0 和16列为段首）的首位。在图形中显示汉字时，一般应在语句末尾用“;”号，以免清掉后部图形。

HPRINT@（行，列）；项目表　　改变高显光标位置，从指定起始位置开始显示。

HPRINT#项目表　　以下显示反相字符（到本语句结束为止）。

HPRINT［字号，字号，……，首字号＼末字号……］　　显示汉字或词组。注意方括号和

"\"号的使用。如用 HPRINT[0\255]即可显示最大容量时全字库内容。

HPRINT 数学表达式(或常数、变量)　　以高显字符显示其值。

HPRINT 字符串(或字符串函数、字符串表达式)　　显示 ASCII 字符(共94个)。

HPRINT "反白英文大写字母"　　显示相对应的英文小写字母。

以上各种显示项目可混合使用,其间以";"","号分隔,规则与 V2.0 同。

(四) 磁带文件操作

在PH系统下,V2.0的磁带操作命令功能仍然有效,但若磁带机已插上遥控插头,则在CLOAD、CRUN、VERIFY命令前应控制软开关走带(CSAVE前还应加上POKE 26624,33)。

PH系统丰富了对原有的T类(BASIC)、B类(机器语言)和D类(数据)文件的管理手段,解决了文件不能指定地址(只能按原记带地址)装载、不能对B文件记带、B文件装入后不能保持静态、D文件数据存贮可靠性差等"老大难"问题。并且增加了磁带文件的拼接或覆盖运行,以及新颖的"调用磁带子程序"、"BASIC程序段记带"等功能。同时还增加了W类(用户汉字库)和"D"类(数据块,与原来的D文件类型码不同,系统能够区别)两种磁带文件类型。PH磁带操作命令,在已联接磁带机遥控线并按下放音键的情况下,均可启动磁带运转,读写结束或出错令磁带自停。

在上述功能的支持下,磁带机这种廉价外设的使用价值有了很大的提高。LASER 310的内存不能容纳的大型应用软件,可以分割为较小的模块,组织成磁带文件集,在主程序控制下自动顺序调入,连续运行,同样能够完成大程序的任务。

1. 写磁带文件命令。

它们的共同点是:走带后自动延时三秒开始输出;参数不能缺省,不用文件名时,须以""号代表;均可用 POKE26624,33:VERIFY命令校对。

(1)SAVET "文件名",段首址,段终址　　将指定的BASIC程序段作为T类文件记带。段终址指段后下个语句的首址,可用LAD函数给出。如以程序首址和终址为段的首、终址,则相当于一条带走带命令的CSAVE。用SAVET可将一个大的BASIC程序分割记带,以后可用LOADT、CALLT、CHAIN和MERGE调用,如文件终址不是程序终址,则不能用CLOAD、CRUN调用。

(2)SAVEB"文件名",首址,末址　　机器语言程序(B类文件)记带。此文件用CLOAD调入将立即运行。

(3)SAVED"文件名",首址,末址　　数据块("D"类文件)记带。其数据结构由用户自行定义。本命令可用于造型表的记带保存,另一个重要用途是屏幕汉字文本及画面记带,例如在高显模式下用SAVED"A",&H4000,&H57FF即可将全屏内容记带,今后可供直接调入显示。只记录某一行、数行汉字或部分画面时,须算准其首末址。

(4)SAVEW"文件名",首字号,末字号　　用户汉字库(W类文件)记带

2. 读磁带文件命令。

它们的共同点是:均有文件类型识别能力,即使不用文件名也不会误装别类文件;不指定装入地址时按文件原记带地址装入,指定地址但省略文件名须用""号代表;命令后跟一#号(如LOADT#)则读带过程中不显示文字提示信息,可避免破坏高分辨率画面(若中途按

BREAK 后需恢复读带提示功能,可用 POKE　30796,0);在显示 WAITING、FONGD 提示信息时,用户可作快进快倒磁带的操作。

(1)LOADT“文件名”,装入首址　　读磁带BASIC程序文件,从指定地址开始存放,并调整各项指针进入可执行状态。与 CLOAD 相似,调入前原来的 BASIC 程序被改变。

(2)LOADB“文件名”,装入首址　　读取磁带机器语言程序文件,不立即执行。

(3)LOADD“文件名”,装入首址　　读取磁带数据块文件。若是高显文本,须在相应的显示模式下用LOADD#读入。高显画面文件一般按原址装载为宜,如改在较高地址显示而超出显示文本区,就可能破坏系统区的内容。当然在确知文件内容和规模的条件下,亦可灵活使用。例如将汉字文本从当前汉字“光标”位置开始装入,显示完毕后又把“光标”指针定位于装载末址之后,便可代替H PRINT 语句的作用,而且不占内存,不过速度较慢。

(4)LOADW“文件名”　　读取磁带汉字库文件,按原址存放。如文件地址不超出现字库范围则更新同址内容后继续执行程序,否则扩充现字库并使变量、串区初始化再继续执行。字号均按现在实际位置定义,不一定与记带时相同。

(5)CALLT“文件名”,暂存首址　　在程序执行中调入磁带 BASIC 子程序并立即执行,遇 RETURN 语句返回主程序。用此方式调用的 BASIC 程序须遵守以下规定:第一,不含 DATA 语句,READ 语句将到主程序中读数;第二,不能在其中嵌套 GOSUB 和 ONERR 等转子语句;第三,GOTO、IF…THEN…ELES、ON…GOTO 等转移语句若向高行号转移,则转到子程序内部,向低行号转移则转入主程序,不再返回;第四,磁带装入的子程序暂存地址须选择执行本程序时不会用到的安全区域。一般可在当前空闲区的低端预留备用的新增变量空间后,开始装入磁带子程序,但须防止高端侵入堆栈区域。

CALLB“文件名”,暂存首址　　在程序执行中调入磁带机器语言子程序并立即执行,遇 RET或RET$_{cc}$指令返回主程序。存放时注意事项同上。

CHAIN“文件名”　　从磁带调人 T 类文件覆盖当前 BASIC 程序,将当前变量区内容传送给新的程序,并立即执行新程序。但用程序中字符串赋值的串变量因地址改变将失效。如原程序中有A $ =“ABCD”,在原程序被覆盖后 A $ 必须重新赋值。因此准备作覆盖运行的程序中应尽量改用 INPUT 和字符串函数的赋值方式。

MERGE“文件名”　　读取磁带 T 类文件,拼接于现有程序之后,行号不改变。若系两个独立程序拼接,应再调用 RENUMBER 程序重编行号。

(五)　其他扩展BASIC功能

1.　十六进制参数。

&Hnnnn　　nnnn为十六进制整型数,可作语句参数使用(要求十进制直接数者除外)。

2.　语句命令。

CALL　入口地址　　调用机器语言子程序。地址用十进制(正)或十六进制直接数。

RST　行号表达式　　将 DATA 数据指针复位到该行首,READ 由此读数。

PST　恢复被 NEW 清除的程序,原程序已部分破坏时可恢复残存部分,但首行可能不正确。如程序区已完全破坏则显示出错信息。

LOMEM　首地址　将 BASIC程序移到指定首址,使其低端成为用户保留区。LOMEM 0 则将程序移回用户区低端(首址7AE9H),清除保留区。此命令对 BASIC 程序无影响,故

可用来将程序在内存中任意移动地址。

AUTO　首行行号，行号增量　自动产生行号进行编程，按 BREAK 终止。

DEL　起始行号-结束行号　删除指定范围内的程序段

DEFDBL　字母或字母范围　将这些字母打头的变量定义为双精度变量。字母间用逗号，字母范围按英文字母顺序，如DEFDBL D-G，W，A。

DEFINT　字母或字母范围　定义整型变量，用法同上。无需再加类型符“%”。

DEFSNG　字母或字母范围　定义单精度变量，用法同上。因 BASIC 每次初始化都将所有变量预定义为单精度，故只在重新定义时需用。

DEFSTR　字母或字母范围　定义字符串变量，用法同上。无需再加类型符“$”。

ERROR　错误编号　模拟指定种类的错误。主要用于调试错误处理子程序。

ON　表达式　GOTO　行号1，行号2……　按表达式值N，转移到后列的第INT(N)个行号的行。若INT(N)＝0或多于后列行号个数，则不转移而顺序执行下行。

ON　表达式　GOSUB　行号1，行号2……　按表达式值调用指定行号入口的子程序。

ONERR　GOTO　行号　规定此后如程序出错则转移到指定行继续执行。

ONERR　GOTO　0　终止出错转移状态(如此时已出错则显示错误信息)。

RANDOM　打开随机数发生器，重播随机数种子。

RESUME　行号　从错误处理子程序返回到指定行继续执行。

RESUME　NEXT　从错误处理子程序返回到出错行的下一行继续执行。

RESUM　0　从错误处理子程序返回原出错行重新执行，但应保证不再出错。

TRON　进入BASIC程序运行跟踪状态，显示执行到的每一个行号。

TROFF　终止运行跟踪状态。

3。函数。

LAD(行号表达式)　行地址函数，其值为该行首址。可用于SAVET等。

CDBL(表达式)　取双精度型值。

CSNG(表达式)　取单精度型值。

ERR　取当前错误类型的代码。

FRE(串变量名)　取串区空闲字节数。

POS(0)　取光标在行中的位置。

STRING$(字数，字符表达式)　取同一字符组成规定长度的字符串。

CINT(表达式)　取整型值。

ERL　取当前出错的行号。

FIX(表达式)　截去小数部分。

MEM　取空闲内存字节数。

VARPTR(变量名)　取变量地址。

(六)　系统扩充选件

1．造型表功能模块(DRAW)。

PH系统磁带上有一个功能扩展模块 DRAW，在PH1.3支持下用 CALLB 或 CRUN调

入后，自动拼装于主模块低端A9F9H～AB78H，此后即可使用造型表绘图功能。加载后字符串区和字库前移，不影响用户BASIC程序。在无DRAW模块时用LOADW记录的磁带文件，在已加载此模块后调入，字库将覆盖DRAW模块，再使用有关命令则会出错。

"造型"是用数字描述的图形，其基本元素是用造型编码表示的绘图向量。每个造型编码为三位二进制数。最高位为1，表示在当前坐标画点，为0表示不画点，后两位表示"移笔"的方向：00向上，01向右，10向下，11向左。以上代码连续组合便可描述出任何图形。造型代码存于内存中，以一个单元的D5～D0六位存放两个代码，D7～D6两位存入00，表示不用，如非00，则表示不画点只移笔的三种代码(相当于001，010，011)。

一个造型的代码表须连续存放在内存的安全区间，组成造型表。造型表的结构如下：第一个字节为表内的造型个数(0～255)，第二字节为0，以下是索引表，每两字节一组，为各组造型代码与造型表首字节间的相对距离(字节数，低位在前，高位在后)。索引表后顺序存放各造型的代码串，每个造型代码串必须以一个零字节表示结束。系统磁带上的"DRAW-DEMO"程序，使用含有一个正方形、一个空心十字形的造型表，各项内容分别以下划线标出：7C6DH——02 00 06 00 09 00 2C 3E 00 2C 2E 3E 3E 3C 2C 00

造型表的编制方法：先在纸上画出图形（因显示时可以放大，故只需画出最小相似形即可），分解为造型向量，逐个列出编码，将编码组合成字节，例如111和010两码可组成00111010B＝3AH＝58，然后按规定格式组成造型表存入。造型表可用SAVED命令记带备用。

对于PH系统，造型表的首字节地址应作为指针存入784EH/784FH两单元（高位在后），以便系统取用。用LOADD由磁带调入一个造型表后，亦须如此。使用造型表绘图的命令是：

(1)SCALE＝表达式　　造型倍率定义，可用来使图形放大。值取1～255，0＝256，为最大值。造型的长度单位为高分辨率显示点的最小间距。显示时每个向量均按设定倍率绘出。

(2)ROT＝表达式　　造型旋转量定义。值取0～63，每单位约相当于0.098弧度。可使造型由原状态旋转规定角度后绘出。

(3)DRAW　造型号　AT X，Y　　从指定原点起绘出指定造型，未定原点则以前一个DRAW或HPLOT绘图的终点为原点。

(4)XDRAW　造型号　AT X，Y　　抹去指定造型，规则同上。

显示时每个点的具体坐标按四舍五入取值，超出屏幕范围时转到相对的一边继续绘制。以7971H为造型显示速度控制单元，存入255显示最快，但屏幕光点闪烁干扰较大，存0时最慢，无干扰，初值为10，用户可视需要设定，其余同APPLE II机，详情可参阅有关资料。

2. 重编行号(RENUMBER)程序。

功能如下：

(1)重编BASIC程序的全部行号，并相应修改下列语句中的目标行号参数：GOTO，IF…THEN…ELSE，GOSUB，RUN，LIST，LLIST，ON…GOTO，ON…GOSUB，ONERR，RESUME，DEL，以及用直接行号作参数的RST语句。

(2)处理300个以下的目标行号参数时不使用用户空间，可处理内存能容纳的最大程序。

(3)遇到错误的行号参数时能将其首数码变为字母(以A～J代替0～9)，以提示修改。

(4)处理过程中如发生内存不够情况，报告溢出时所处理到的行号，以便删节或分割后

重编；

（5）两个以上程序拼接后（行号不呈升序）重编，各自的内部转移目标不会错乱。

（6）产生过大行号（≥60000）时能自动加以调整。

（7）速度快，安全。

用法：在PH1.3系统支持下，将文件用LOADB命令调入内存（与BASIC程序无冲突）后，即可使用。重编命令的格式为“RENUM 行号初值，行号增量”。本程序以显示区存放并为第一工作缓冲区，故工作时屏幕出现点线干扰以至图案字符，均属正常，工作完毕自动清屏。如执行过高、中分辨率显示命令，则需重新调入本程序方能使用RENUM功能。

3. 字库建立和编辑（CREATE/EDIT，简称C/E）程序。

本程序专供字库建立及编辑时使用，因它由BASIC与机器语言混合编制，并使用用户程序区低端，故一般不宜在已有用户程序时调用。

本文件前面录有使用说明显示程序“C/E REM”及其汉字库。运行“C/E REM”程序时，将自动调入字库，然后显示说明文本。说明文字共三页，按空格键翻页，最后一页按其他键即开始装入“CREATE/EDIT”程序。说明字库仍保留可用。PH1.3B的“C/E REM”程序是采取直接调带显示的方法，不用字库。

C/E程序启动后，将要求用户给定所需最大字号（加1＝全库总字数），以申请字库空间，内存中已有字库时可以扩大或缩小其容量，扩大不得超出256字，缩小时字库低端（小字号）将被清除，字号相应改变。在原库规模下进行修改，不再给字号，只按RETURN键即可。此后将显示字库内容，空闲位置为空格稿笺式样。屏幕左上角出现一个反白的光标，表示准备接受操作命令。屏幕末行为字库操作窗口，显示有关状态及提示信息。

操作前应接上磁带机遥控线，将字库磁带放入，倒进带至目标区段之前，按下放音键。

（1）操作命令：

RETURN键——清屏后调入磁带当前的基本字库文件（不指定文件名），此文件内容只在屏幕上显示，以供取字，不直接进入字库区。

“P”——调入磁带当前的用户字库文件（不指定文件名），装入字库区，到窗口的“LOADW”提示消失、光标重新出现时，装载完成。屏上原显示内容不变，仍可取用。

↑↓←→键（不按CTRL）——上、下、左、右移动光标，光标所在的字可供取入字库。

数字键——指定字库的操作（置入、代换、插入或删除）目标字号，设定字库指针。

“-”——清屏后字库分页列表显示（字数少于177作一页，否则0～127为第一页，其余为第二页，轮番显示）以供编辑。

“:”——进入字库编辑状态，以下可执行编辑命令。

（2）进入字库编辑状态后，窗口显示当前字库指针值及当前字（反相显示），作为操作目标。以下可使用的编辑命令是：

“:”——字库指针值加1，操作目标指向下一字号，连按则向前扫描字库。

“*”（SHIFT＋:）——指针值减1，指向上一字号，连按则向后扫描字库。

“>”（SHIFT＋.）——将光标所在的字送入字库指针当前所指的位置，并显示于窗口（正相），原有字被取代。

“L”——将光标所在汉字插入字库指针当前所指位置，其后各字依次后移（末字被删），字

库指针值加1。

“;”——将字库指针所指汉字删除,其后各字依次前移,末字补空。

其它键——退出编辑状态,回到取字阶段。

(3)按BREAK键,退出本程序。用户字库记带用SAVEW命令。

(4)退出后需再次进入本程序继续工作,RUN后只按RETURN键,以免清库,用HPLOT:GOTO 47重新进入,原来屏上内容除首行汉字外仍可重现并供使用。读带出错时,也可如此操作。

二、设计的基本方法与技巧

扩充系统设计面对的主要问题有四个:第一,需要新增加不少程序模块,以支持高分辨率绘图和汉字显示等功能;第二,必须解决与基本系统和原已配置的扩充系统(如DOS)的兼容问题;第三,因汉字点阵数据占内存多,系统本身对内存的使用必须尽可能地节省,为用户保留必要的自由空间;第四,以磁带机作为基本外存贮器,要求增强磁带文件操作功能,为小内存运行较大的程序提供支持。因此,设计的难度较大。

作者研制扩充系统时,除依靠原来的基本系统资源外,有的功能虽是模拟别的系统,但算法、程序都是自行独立设计,并非移植;还有不少独创功能是“BASIC大全”也未曾有过的。由于个人才能、精力和时间的限制,虽经反复修改,但还远谈不上达到优化和完美的程度,很多地方都有明显的斧凿之痕。现在将它原原本本地和盘托出,主要是用以证明扩充和改造一个系统程序是完全可能的,以及提供基本的方法和技巧。有兴趣的读者不妨在此基础上进一步改进提高,使之更好地为你所用。同时也希望使用其他学习机的用户由这个实例得到鼓励,萌生出“仿此办理”的打算。

(一) 高分辨率显示及绘图

PH系统只使用了MC6847的RG6模式(256×192),我们将其称为“高分辨率”模式,简称“高显”。而V2.0的MODE(1),则改称“中分辨率模式”。PH的高显功能,包括高显清屏,高显绘图,高显汉字和ASCII码,高显图型表等。

1. 显示分辨率的转换。

硬件改造后,只要向输出锁存器的Q7,Q6和Q3写入一定的数据便能实现显示分辨率的转换。此三位数据是1,1,1为高分辨率,0,0,1为中分辨率,0,0,0或1,1,0为低分辨率。有一点需要强调,输出锁存器Q1状态只决定CPU访问4000H~6FFFH空间时的实际目标器件,并不影响显示模式。只改变模式控制位数据时,必须保护其余各位的原状。这可用适当的掩码与原输出数据的副本进行逻辑运算来达到。

在分辨率改变时,显示文本区的范围将随之变化,而且同一代码在不同模式下的显示图像是不一样的。当显示区扩大后,原使用的低端部分的代码仍将显示在屏幕上端,而且变成杂乱的图像,影响显示画面。V2.0在模式转换时就具有自动部分清屏功能。这一功能。在PH系统下由高显回到低显文本模式时仍可发生作用。

每次进入高显状态时,取输出锁存器副本(783BH)测试原来模式,若是由低显进入,则清除7000H~71FFFH,从中显进入则清除7000H~77FFH。这以清屏标志值1或2表示(3为清全屏),由清屏执行程序按码分别进行。

2. 高分辨率显示的时序安排。

根据LASER 310硬系统的结构，同中、低分辨率显示一样，必须解决CPU和显示发生器在访问VRAM上的冲突。也就是要将它们的访问安排为分时进行，CPU访问VRAM最好利用视频的场消隐期（以及上下边框期，下同）。

为此，高显功能需要分为语句解释和显示执行两步来完成。在场扫描期间，BASIC解释程序对语句进行解释，将需要输出显示的内容存放到输出缓冲区，并置标志表示有信息等待显示。高显执行程序是"嫁接"在V2.0监控程序的用户出口上的，初始化时就在787DH设置了入口向量。当场消隐期到来，CPU响应$\overline{INT}$请求进入中断响应后，将首先访问PH系统的监控扩充部分，依次检测清屏标志、绘图标志和汉字标志。若发现某标志有效，就转入相应的执行程序输出显示。高显标志全都无效时，再执行V2.0的监控程序。

高显的特点是显示信息量大，低显的一个字符用一字节数据，一个汉字用32字节数据；而且处理过程较为复杂。因此，显示需时较多。如前所述，场消隐期加下边框共历时7ms，若显示信息过多，就会延长到6847访存时期内，造成对屏显的干扰。因而必须对每次的显示信息量进行控制。经实验，每次中断显示汉字控制为一个（每秒约50字），绘图控制在50个点以内。这样基本上消除了屏幕上的点线干扰。为了简化程序，清屏和屏幕卷动等未加控制，干扰仍少量存在，但已无伤大雅，可以忍受。

3. 屏幕显示数据结构。

RG6模式下，全屏分为192个显示行。每显示行对应于VRAM的32个字节。每位映射为屏幕上的一个像素，因而水平方向为32×8=256列。每个像素数据值可取0或1。在COLOR，0时，为浅绿/深绿二色；COLOR，1时，为白/黑色。

VRAM中显示数据由低地址到高地址的顺序，对应于屏幕显示位置的从左到右和从上到下。由屏上的显示行、列位置换算为VRAM单元地址的公式是

地址＝4000H＋行数×20H＋INT（列数/80H）

一个VRAM单元中各位数据对应的屏幕显示顺序却相反，高位在左，低位在右。由显示的列数计算它在本单元中位数的公式是

位数＝07H－〔列数－INT（列数/08H）×08H〕

4. 画（抹、测）点的方法。

图形是借取值相反的像素间的明暗对比显示出来的。以0值像素为背景时，可用1值像素绘图；反之，也可用1值像素为背景，0值像素绘图。同样，若使一个像素数据变得同周围的背景像素一致，便是"抹"掉了一个图形点。可见画点或抹点（以及对图形点的测试）都应是位操作。

以HL间接寻址的位操作指令，都由两字节构成，第一字节都是CBH，第二字节的D5D4D3是位地址码，D7D6…D2D1D0是决定操作类型的操作码。若对（HL）的D0进行操作，SET指令是11000110（C6H），RES指令是10000110（86H），BIT指令是01000110（46H）。可利用它们作为指令码的"框架"，"填入"不同的地址码来实现位操作。我们采用了一个公共程序来实现画、抹、测点的功能。首先按行、列数求出目标单元地址，放入HL；求出位数并放入累加器A的D5～D3位置成为位码（其余位为0）。用指令码框架之一与位码进行OR运算，便合成了所需的指令，执行此指令即实现规定的操作。详见PH程序解说。

5. 画(抹)线的方法。

高分辨率画线实际就是连续画点。程序只给出起点和终点的坐标，算出中间每个点的坐标是绘图程序的主要任务。用函数方法计算虽然比较简易，但调用函数程序需时较多，绘图速度慢，所以还是直接计算的方式较为适宜。

我们以画线的X、Y两变量中步数较大(或相等之一)的为“主变量”，另一个为“从变量”。显然，主变量的步长可以是1，从变量的步长必然是纯小数。以主变量的步数作为连续画点的循环变量，每次主变量增1，从变量加一个步长值，从而得到下一点的坐标。为了简化计算，使用二进制定点数，以寄存器对的低八位存放从变量的小数部分，高八位存放整数部分；作为坐标值时只取整数，小数部分按当前值四舍五入。由于舍入有误差，同样两点间的线段，从相反方向画来不一定重合。所以抹去一条线段时必须按画线的同一方向进行。

一个点是没有方向性的，坐标都是正值。画线却有方向性。X以自左至右为正向，Y以自上至下为正向；反之则为负向。正向变量按步长增量，负向变量按步长减量。所以变量的步长应是有符号数。符号在取参数后通过初值与终值的比较获得。主变量的符号存在工作单元里，每次取出测试以决定是加1还是减1。从变步长是纯小数，以高八位的00H或FFH表示符号，实际上就是变成补码形式；与当前变量值相加后只可能是正数，所得的结果就是原码。

从速度考虑，最好在语句解释阶段就把各点坐标一一求出，存于缓冲区，中断期间迅速输出。但因扩充系统面临内存紧张的局面，不宜采取这种方式，只是先将步长等参量算出，存于工作区，由显示执行程序进行累加运算后输出。加上运算所费的时间，所以每次中断期间就只能处理50个点了。

(二) 汉字显示

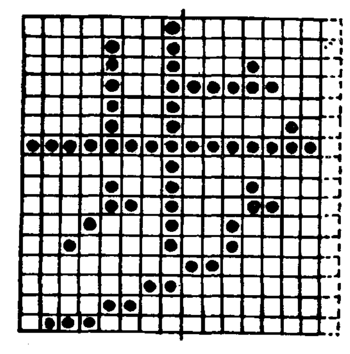
计算机的汉字，实际上是作为特定的图形单位来处理的。因而汉字显示和高分辨率绘图实质相同。主要区别在于，汉字显示是将字库中的现成数据传送到显示区，系统只需确定目标位置，不必考虑“图形”的内容；同时每次不是处理一个点，而是若干个目标地址不连续的字节组合。

1. 汉字结构。

汉字作为一种特殊的图形，其数据结构分为点阵型和向量型两类，一般采用点阵结构。点阵汉字是由若干个点(如16×16，24×24等)组成的两维矩阵。每一维的元素越多，笔划的分辨率就越高，字形也就越清晰美观。现在有的文字处理系统已使用128×128的点阵汉字。但矩阵规模越大，需要存贮器容量越多，不但字库空间增大，而且需要更大的显示RAM和更高分辨率的显示器件，也就会增加系统的造价。但是因为很多汉字笔划复杂，小于16×16的点阵将无法表现。所以，普及机均采用这一规格的点阵汉字。

为了使汉字信息标准化，国家制定了《信息交换用汉字15×16点阵字模集》(GB 5199.1—85)和《信息交换用汉字15×16点阵字模数据集》(GB 5199.2—85)两项标准。标准规定，以横向15格，竖向16格的等距离栅格对构成点阵字模的点进行定位。图10-1是作为例子的汉字“步”的字模图。“数据集”规定字符的点阵数据为16×16，计256个二进制位，合32个字节。其中第16列全为空格，作为字与字间的间隔。同时，横向用15个点也较为适应汉字左右对称的特点。点阵字模数据的排列顺序是：第0行 0字节，1字节；第二行 2字节，3字节……第15行 30字节，31字节。因此，“步”字的点阵数据是：01,00;09,00;09,10;09,F8;09,

00;09,04;FF,FE;01,00;09,10;0D,18;11,20;21,20; 00,C0;03,00;0C,00;70,00。


图 10-1 汉字点阵字模

标准字模采用书版宋体字型，除基本笔划外，还有装饰点。但因点阵规模的限制，有部分字笔划无法分离，而采取了简化结构。

国家标准字符集提供的字符共 7451 个，分为 87 个区。1～9区有外文及其他图形字符688个，16～55 区有一级（常用）汉字 3755 个，56～87 区有二级（非常用）汉字 3008 个。每区最多包含 94 个字符（位号01～94）。用区号和位号可以指定一个特定的汉字或字符，称为“区位码”，它是各种汉字编码方案的基础。

2. 汉字点阵信息源——汉字库。

汉字功能必须有汉字点阵数据库的支持。通常使用的汉字库有硬字库和磁盘字库两种。硬字库是将点阵数据固化在 ROM 中，装在机内或外接的插卡上。硬字库调用方便快速，可以不占实存空间，但成本较高。国内的其他几个 LASER 310汉字系统都使用硬字库。磁盘字库存贮介质虽然远低于硬字库，但要求配置较为昂贵的驱动器，而且数据装入内存也需要可观的存贮空间，对 LASER 310 来说可行性不高。PH 系统按照“最低配置”的设计思想，选择了国内外尚无先例的“磁带软汉字库”方案。

按国标字符集制成的硬字库 ROM 规模为 2M bit(256KB)。在一般应用上，一级字库已基本可以胜任，但仍需 3755×32≈120KB 数据量。将这些数据分区记录在磁带上，正好是运转 30 分钟的一面磁带。

磁带记录的数据既不可能全部装入内存，又受普通磁带机非智能性的限制不能随机提取数据，所以只能采取“用户字库”作为它同用户程序之间的“接口”。磁带字库作为基本的数据资源，由用户按特定程序具体需要的汉字，从中选取建立用户字库，以支持程序的汉字显示功能，解决小内存下使用汉字的矛盾。这就决定了磁带软汉字只是作为软件开发的一种辅助性工具，而不可能支持大批量的文字处理工作。

6847的 ASCII 字符点阵，本来就是高分辨率的，但可惜不能调出利用。在高显状态下，需要与汉字库相似的 ASCII 字库。国标字库中虽含有 ASCII 字符，但都是“全角”字符，占一个汉字位置，不切合实用。所以系统增设 8 × 9 的 ASCII 点阵字库，一次性装入扩充的 VRAM 内 5CA0H 开始的空间，支持显示。

磁带上的汉字数据必须装入内存才能加以选择和取入用户字库。若用户字库为 256 个汉字，磁带字库每次也取入 256 个汉字进行选择，这就共需 512×32=16KB 空间，占了 LASER 310 基本系统的全部主存贮器，显然是不行的。若每次装入供选用的字数过少，又会使操作不便的缺点愈加突出。作者为此伤了很久的脑筋，才想出直接利用显示文本区的方法。高显的显示文本区 6KB，可供存放 192 个汉字的数据。如果将汉字点阵按显示的格式放入，正好便得到一张直观的字库表。磁带数据直接装入显示 RAM，丝毫不占用户空间，可谓一举两得。有了屏幕上的字库表，就不需要任何编码体系，直接以键控光标扫描屏幕的方式，便可把需要的汉字取入用户字库了。每屏装载两个区的国标汉字后，还有四个字的空位，正好作为建库程

序工作的提示窗口。平心而论，以上这种看来颇似“精心”的安排，相当程度上应归功于“机遇”——如果没有了这些巧合的条件，PH系统也许就是另外的面貌了。

屏显字库方式要求磁带软字库不能采取标准字库的数据结构。因为磁带数据是按存贮地址顺序装载的。所以写入磁带的汉字点阵数据应以一行（16字）为一组，按以下顺序（括号外为字号，内为字节号）：0(0,1)，2(0,1)……15(0,1)，0(2,3)，1(2,3)…………0(14,15)……15(14,15)。然后记录下一行。磁带字库装入时，也不是按字显出，而是依次用16条高分辨率显示线“凑”成整行汉字。建库程序取入光标指定的汉字时，再按标准数据格式的字节顺序存入用户字库。

3．汉字及ASCII字符的显示。

PH系统的HPRINT语句，具有同基本PRINT语句相似的语法成分和规则。所不同者，一是汉字用“〔字号〕”，词组用〔字号＼字号〕表示，二是反白汉字或ASCII字符用#号标出，三是借用原来的反白英文大写字母表示小写字母。

语句解释程序在识别各种语法成分后，便把相应的显示码送入输出缓冲区，在中断期间由汉字显示执行程序输出显示。由于高、中、低分辨率显示不可能同时进行，所以它们可共用同一的输出缓冲区（7AB2H～7AD1H）及其指针（7AB0/7AB1H）和计数器（7AAFH）。为了使双方的数据不致混淆，HPRINT语句输出汉字以前要先检测缓冲区计数器是否为0。若输出缓冲区不空，则循环等待到原有低显字符已输出完毕，方开始输出汉字显示码。由汉字显示回到低显文本模式前，也要等待缓冲区内容显示完毕，使指针复位，以“完璧归赵”。

HPRINT输出的显示码如下：

00H　字号　　正相汉字。

01H　字号　　反相汉字。

20H～7FH　　正相ASC字符。

A0H～FFH　　反相ASCII字符。

0DH　　　　换行。

0FH　　　　跳到下一段显示（每行分为两段）。

此外的数据均无效，不作任何显示。

汉字显示的主要方法是：

(1)显示地址计算。为了给显示字符定位，将屏幕分为12行，32列。每列可显示一个ASCII字符，两列显示一个汉字。用行列指针而不是实际地址来指定显示字符的位置。行数为0～11，列数为0～31。向显示区输出汉字点阵数据时，计算实际目标地址的公式是：

首字节显示地址＝4000H＋行数×20H×10H＋列数。

以后各字节的显示地址，奇数字节为前一偶数字节地址加1，偶数字节为前一奇数字节地址加31。如字首址（0字节地址）4000H，1字节为4001H，2字节为4032H，3字节为4033H……

ASCII字符的首字节地址计算方法相同，后续字节地址每次增值32。因为字符高度只有9个显示行，所以开始的6字节和最后的1字节均不送入字符点阵数据，而是送入00H，使之与汉字显示协调。

(2)字库地址计算。汉字字号就是在字库中的序号，一个汉字点阵数据的首址＝字库首址

+字号×20H。

ASCII字符点阵数据首址=字库首址+(ASCII码-20H)×09H。

(3)相位处理。为了丰富汉字显示，设置了正反相显示方式。正相就是笔划点与屏幕基相相反，空白点与屏幕基相相同。反相则各点都与此相反。可见，显示相位是相对的，正相屏幕背景上的正相字符，与反相屏幕背景上的反相字符相同。具体的实现方法就是在每一字节送显示地址前，将它同屏幕基相标志和显示相标志（都以00H表示正相，FFH表示反相）连续作两次 XOR 运算，即得到所需相位的数据。

(4)换行与卷屏。汉字显示的一个关键技术问题是换行和屏幕卷动的实现。

换行分控制换行和自然换行两种。前者根据 HPRINT 语句的结束符，结束本行，准备开始下行。后者在行满后自动转入下行。所以每显示一个字符，都要测试是否超出32列的范围。在汉字和 ASCII 字符混合显示的行中，有可能到了最后一列却需要显示汉字。这就会将一个汉字劈为两半。测试出这种情况应在这一列补上一个 ASCII 的空格，而把这个汉字显示在下行行首。

换行除了改变汉字"光标"——行列指针外，一项重要工作是将行的后部清为空格。空格数由行长减显示字符数求得。跳段显示的方法与此相仿。

当换行发生在屏幕最底部的一行时，就产生了屏幕卷动问题。为了简化操作，我们采取数据块传送指令，将屏幕第二行至末行的全部数据传送到显示区首，再将末行清空，作为当前行。由于传送数据量大，指令周期长，总的时间大大超出视频的场消隐期，所以屏幕上出现点线干扰，这是它的缺点。

（三） 磁带文件操作

为了在低配置、小内存的环境下运行汉字程序，必须充分发挥磁带机的作用。V2.0 的磁带操作命令功能较为简单，而且还有一些明显的缺陷。剖析它的磁带读写功能程序后，发现只须在其基础上作一些修改，就能显著提高工作能力。考虑到修改之处并不多，单独形成一个程序文本未免浪费空间，就采取了"临时修改"的方案。在每次调用读写磁带功能程序时，将它传送到输入缓冲区，进行修改后立即调用，用后废弃，再用重来。因为用于修改程序显著短于读写程序本身，可以节省内存。虽然时间开销增加了"很多"，然而对于磁带机这样的低速设备，仍在可忽略的范围。事实上用户感觉不到这种方式比原来的直接调用来得慢。

磁带读写功能的主要改进有以下内容：

1. 文件类型的增加。

为了解决 V2.0 的 PRINT# 和 INPUT# 两条存取磁带数据文件工作不可靠的问题和满足存取用户汉字库的需要，增加了数据块("D"型)和汉字库("W"型)两种文件类型。所谓文件类型，关键就在一个识别码。V2.0 规定 BASIC 文件为 F0H，机器语言文件为 F1H，数据文件为 F2H。它们是在文件之前记录到磁带上的。读带程序从磁带上读到类型后，将它作为一个地址的低八位，于是指向一个数据表中的特定单元，此单元所存代码经运算便产生出提示行首显示的"T""B"和"D"字符——类型标志。根据这一原理，不必修改有关程序，只需提供新的类型码，便能创设新的文件类型。为了使显示的类型标志字符能反映类型的属性，我们选择了F3H为汉字库文件类型码，因为由它构成的地址正好指向一个"W"字符码，可表示 Word。选择FFH为"数据块文件"类型码，因为由它构成的地址指向一个"D"字符码，用以表示

"Data";它虽与 V2.0 的 F2H 文件类型显示同一标志字符,但因类型码不同,系统能够区分。

V2.0 虽有三种文件类型,但它的 CLOAD 程序在装载时却是把 T、B 两类混为一谈,只是不理睬 D 类文件。在装入类型码后进行测试的指令是:

```
3673-FEF2     CP F2H        ;是 D 类型码吗?
3675-28F6     JR NZ,366D    ;是D类文件则重新搜索。
```

我们把这两条指令改动一下,3674H 填入指定装入的文件类型码(这里以 nn 代表),3675H 改为 20H(JR NZ,DJS指令操作码),指令就变为:

```
3673-FEnn     CP nn         ;是指定类型码吗?
3675-20F6     JR NZ,366D    ;不是指定的文件类型则重新搜索。
```

这样一改,程序便能区分所有的类型,不理睬非指定类型的文件了。

2. 指定装入地址。

V2.0 的记带程序将文件的首、末地址记录在文本之前。读带程序取入后就将文本从原来首址位置开始装载。这时,首址在 DE 中。此后有一段程序测试 I/O 控制字,以确定是不是 VERIFY 调用。在扩充程序中这并不需要,因而可以把这里改为控制装入首址的程序段。

为了简化修改程序,预先准备了一个指令块(PH 扩充程序的 B32BH~B336H)。根据是否指定装载首址选择 B32BH 或 B32FH 开始的 8 个字节指令移入上述空间。选择前者时,指令是LD DE,(79D7)和 LD (79D7),DE,便以预贮在(79D7H)中的装入首址取代了原来 DE 的内容;选择后者,指令是LD(79D7),DE,后为 4 条空操作指令,DE 仍保持不变。

3. 结束处理的修改。

V2.0 读带程序的结束处理,B 型文件是立即转入执行,T 型文件是使 BASIC 指针初始化后回到等待输入阶段,实际使用时均有所不便。为满足对文件操作的多方面需要,我们把原来的结束处理段截掉,用 RET 返回调用的工作程序。由扩充的各种磁带操作命令工作程序分别采取不同的结束方式。

4. BASIC 程序段文件。

为了使较长的程序能够分段记带,今后调入单独运行或进行拼接、覆盖运行,以适应小内存的条件,我们创设了"BASIC 程序段"文件。这种文件在记录手段上并无特别之处,给出程序段的首末址即可用记带公共程序完成。比较独特的是装入后的处理。因为程序段是不带程序结束标志的,装入后修改行指针就会造成程序迷路而"死机"。所以,SAVET 工作程序在文件装载结束时要测试程序结束标志,若发现没有即予补上,然后方转入修改行指针阶段。

5. 用户字库文件。

用户字库文件与别种文件的一个不同点是它不能任意指定装入地址,而只能按原址存放,以避免地址给定不当对系统工作产生破坏。因为字库的顶上是字符串区和堆栈,若新装入的字库大于原来的字库,就可能覆盖堆栈,造成死机。所以,LOADW 程序在读带前要记存当前堆栈指针,并在内存高端设置一个临时堆栈以支持读带操作,同时将原堆栈的最近一个断点转存起来。文件装入后,若原来的堆栈并未破坏,就可原样恢复,继续解释执行程序,不中断运行。若堆栈已被破坏,就在新的软终端之上建立新栈,并使 BASIC 指针初始化。这时虽不中断程序的运行,但已丢失原有的各种中间数据。



三、PH1.3程序解说

为了充分利用内存空间，PH系统程序的各个基本模块是分散存放的。主模块存放于内存高端，各型文本的实际地址依所适应的内存规模而异，下面的解说仅以基本型(18K 基本配置)为例。ASCII 字库和部分扩充语句工作程序的入口地址表，存放在扩充的 VRAM 高端的 2KB 未用空间。此外，一些小的模块则利用了 V2.0 的系统工作区空闲部分。系统软件中的一些非常用模块，作为“选件”，采取装配式结构，由用户根据需要选择安装。

(一) 系统参数和工作单元

PH 系统除使用 V2.0 的系统参数外，又增加了自己的一些标志、指针变量和系统工作单元，大都是“见缝插针”地安排在原来的系统工作区内。主要有：

7803H～7B05H	扩充保留词识别程序（即RST 10H）的入口向量（JP AD2BH）。
7B10H～7B12H	代替7803H作为V2.0“取下一字符”子程序入口向量（JP 1D78H）。
782AH	图形相位标志（00H—正相，FFH—反相）。
782B／782CH	前级系统（V2.0、DOS或MONITOR）的RST 10H程序入口地址寄存器。
7847H	清屏标志（00H—不清屏，01H—清除4000H～41FFH区段，02H—清除4000H～47FFH区段，03H—清除4000H～57FFH全屏显示区）。
7848H	输入相位标志（00H—正相，FFH—反相）。
7849H	显示相位标志（00H—正相，FFH—反相）。
784AH	屏幕基相标志（00H—深绿／黑，FFH—浅绿／白）。
7850H～7859H	HPLOT工作区
7850H	X初值及当前值。
7851H	Y初值及当前值。
7852H	X终值。
7853H	Y终值。
7854H	主变量标志（00H—X为主变量，FFH—Y为主变量）。
7855H	主增量符号（00H—正，FFH—负）。
7856／7857H	从变步长（16位补码表示的8位纯小数）。
7858／7859H	从变量当前值（小数部分／整数部分）。
785AH～785BH	HPOINT使用的寄存器（存放X、Y参数值）。
785CH	HPLOT开关（00H—不绘图，FFH—绘图）。
789FH	HPLOT点计数器（00H～FFH，00H＝256）。
7953／7954H	扩充代号（“小代号”——00H～12H）相对地址表首址。
796B／796CH	简单变量字节数寄存器。
79A2／79A3H	可供用户设置使用的 & 入口。
79AA／79ABH	用户汉字库终端指针。
79AFH	汉字显示标志（00H—不显示汉字，FFH—显示汉字）。

79B0／79B1H　　汉字显示的行列位置指针。
79C8／79C9H　　数据块传送暂存区首址指针。
79D4／79D5H　　数据块传送暂存区终址指针。
79D7／79D8H　　磁带文件装入首址指针。
79DD／79DEH　　磁带文件装入终址寄存器。
79E2H　　用户汉字库字数指针（最大字号）。
79E3／79E4H　　用户汉字库首址指针。

（二）　扩充保留词表检索程序(磁带文件名“PH-2”)

```
785D-  117AAB   LD DE,AB7A    ；DE＝扩充保留词表首址。
7860-  014001   LD BC,0140    ；BC＝表的长度（检索范围）。
7863-  F7       RST 30        ；取待识别字符串首字符。
7864-  EB       EX DE,HL      ；HL＝表首址，DE＝待识别字符串首址。
7865-  D620     SUB 20        ；将首字符变码，同一般字符码相区别。
7867-  EDB1     CPIR          ；在表中检索首字符。
7869-  C0       RET NZ        ；若表中无此首字符则返回调用的程序。
786A-  D5       PUSH DE       ；发现相同的首字符后暂存当前字符串首址。
786B-  13       INC DE        ；指向下一字符。
786C-  1A       LD A,(DE)     ；将下个字符码取入A。
786D-、BE       CP (HL)       ；与表中的下一字节相比较。
786E-  2804     JR Z,7874     ；若二者相同，转去测试保留词是否结束。
7870-  D1       POP DE        ；若二者不同则取回本字符串首址。
7871-  1A       LD A,(DE)     ；首字符重新取入A。
7872-  18F1     JR 7865       ；转去继续往下检索。
7874-  23       INC HL        ；前面字符相同时指向表中下一字节。
7875-  7E       LD A,(HL)     ；下一字节取入A。
7876-  B7       OR A          ；测试是否保留词结尾标志。
7877-  20F2     JR NZ,786B    ；不是则转去与当前字符串下一字符比较。
7879-  23       IBC HL        ；字符串与保留词完全相同时指向后边代号。
787A-  7E       LD A,(HL)     ；代号取入A。
787B-  C1       POP BC        ；清除堆栈中的本字符串首址。
787C-  C9       RET           ；返回（此时Z标志置位）。
```

（三）　“磁盘BASIC出口”利用模块(磁带文件名“PH-3”)

```
7952-  C9       RET          ; 返回。
7953-  7A                    ; 扩充代号相对地址表首址（7A5CH）。
7954-  5C
7955-  F7       RST 30       ; 【LAD函数子程序】取下一字符。
7956-  CF       RST 08       ; 核实左括号。
7957-  28CD                  ; （7957-  左括号码。）
7959-  02                    ; （7958-  CALL 2B02H; 取参数表达式值。）
795A-  2B
795B-  E5       PUSH HL      ; 存扫描指针值。
795C-  CD2C1B   CALL 1B2C    ; 在程序语句表中搜索指定的行号。
795F-  D2D91E   JP NC,1ED9   ; 无此行号则报告出错。
7962-  60       LD H,B       ; 目标行的首址放入HL。
7963-  69       LD L,C
7964-  CD9A0A   CALL 0A9A    ; 作为函数值存入WRA1。
7967-  E1       POP HL       ; 恢复扫描指针值。
7968-  C32F25   JP 252F      ; 转去核实右括号后返回。
796B-  00       NOP
796C-  00       NOP
796D-  00       NOP
796E-  00       NOP
796F-  00       NOP
7970-  00       NOP
7971-  00       NOP
7972-  00       NOP
7973-  C3711F   JP 1F71      ; 【工作子程序入口向量表】ONERR
7976-  C3EDB2   JP B2ED      ; HPOINT
7979-  C3BCB6   JP B6BC      ; RENUM
797C-  C34A1E   JP 1E4A      ; （以下为DRAW模块保留4个命令出口）DRAW。
797F-  C34A1E   JP 1E4A      ; XDRAW
7982-  C34A1E   JP 1E4A      ; SCALE=
7985-  C34A1E   JP 1E4A      ; ROT=
7988-  C3F71D   JP 1DF7      ; TRON
798B-  C3F81D   JP 1DF8      ; TROFF
798E-  C3F41F   JP 1FF4      ; ERROR
```

```
7991-  C3001E    JP 1E00        ; DEFSTR
7994-  E5        PUSH HL        ;【& 程序】存扫描指针值（&字符地址）。
7995-  F7        RST 30         ; 取下一字符。
7996-  FE48      CP 48          ; 是不是“H”（十六进制数标识符）？
7998-  2004      JR NZ,799E     ; 不是十六进制数则跳转。
799A-  C1        POP BC         ; 清除堆栈所存的&字符地址。
799B-  C350B2    JP B250        ; 转入取十六进制数子程序。
799E-  E1        POP HL         ; 不是十六进制数时恢复扫描指针值。
799F-  7E        LD A,(HL)      ; 将&字符码重新取入A。
79A0-  B7        OR A           ; 置标志。
79A1-  C32D01    JP 012D        ; 显示“磁盘命令出错”（可供装入向量）。
79A4-  00        NOP
79A5-  00        NOP
```

（四） 扩充代号相对地址表及ASCII字库(磁带文件名“PH-4”)

1. 扩充代号及其工作程序地址。

由于扩充的 BASIC 保留词较多，≥80H 的代号已不够安排，因而增加了一些小于 20H的代号，称为“小代号”。它们是：

HGR(00H),HPRINT(01H),HPLOT(02H),SAVET(03H),SAVEB(04H),SAVED(05H), SAVEW(06H), LOADT(07H), LOADB(08H), LOADD(09H), LOADW(0AH),CALLT(0BH),CALLB(0CH),CALL(0DH),CHAIN(0EH),MERGE(0FH),RST(10H),PST(11H),LOMEM(12H)。

系统从扩充保留词表取得代号后加以测试，若是小代号则到相对地址表中取得一个 16 位的相对地址，再加上基址（AD1AH），就是工作子程序的入口。相对地址表（5C7AH～5C9FH)依次存贮 19 组地址，均按低/高字节顺序排列。它们是：

05A6H 063FH 01D0H 03E5H 03EBH 03EEH 044CH 04A9H 04C1H 04D5H 04D9H 056DH 055DH 046CH 0342H 0395H 093AH 008DH 00B8H

2. ASCII字库(5CA0H～5FFFH，略)。

（五） 主模块(磁带文件名“PH-5”)

1. “汉”字点阵数据(AB59H～AB78H)。

当汉字库字数指针为 0 时，字库中可有(并且应该有)一个汉字(0号字)。为了便于用户进行试验性的汉字显示，系统中含有一个“汉”字的点阵数据，可用 HPRINT[0]显示。

2. 扩充保留词表。

PH 的扩充保留词表存放在系统主程序之前。它采取散列表的结构。每个保留词字母的以 ASCII 码依次存放，首字符的 ASCII 码减去 20H 变为小于 40H 的代码，以便搜索时识别。保留词代码串以 00H 作为结束标志，其后紧跟这个保留词的代号。接着是下一组保留词

的数据。

扩充保留词表

HGR	28 47 52 00 00	HPRINT	28 B2 00 01
HPLOT	28 50 4C 4F 54 00 02	HPOINT	28 C6 00 C7
SAVET	33 41 56 45 54 00 03	SAVEB	33 41 56 45 42 00 04
SAVED	33 41 56 45 44 00 05	SAVEW	33 41 56 45 57 00 06
LOADT	2C 4F 41 44 54 00 07	LOADB	2C 4F 41 44 42 00 08
LOADD	2C 4F 41 44 44 00 09	LOADW	2C 4F 41 44 57 00 0A
CALLT	23 41 4C 4C 54 00 0B	CALLB	23 41 4C 4C 42 00 0C
CALL	23 41 4C 4C 00 0D	CHAIN	23 48 41 49 4E 00 0E
MERGE	2D 45 52 47 45 00 0F	RST	32 53 54 00 10
PST	30 53 54 00 11	LOMEM	2C 4F 4D 45 4D 00 12
LAD	2C 41 44 00 BE	RENUM	32 45 4E 55 4D 00 A2
DRAW	24 52 41 57 00 A3	XDRAW	38 44 52 41 57 00 A4
SCALE=	33 43 41 4C 45 D5 00 A5	ROT=	32 4F 54 D5 00 A6
TRON	34 52 4F 4E 00 A7	TROFF	34 52 4F 46 46 00 A8
ERROR	25 52 52 D3 00 A9	DEFSTR	24 45 46 53 54 52 00 AA
DEFDBL	24 45 46 44 42 4C 00 9B	DEFINT	24 45 46 D8 00 99
DEFSGN	24 45 46 53 4E 47 00 9A	AUTO	21 55 BD 00 B7 00
DEL	24 45 4C 00 B6	ONERR	2F 4E 45 52 52 00 85
ON	2F 4E 00 A1	RESUME	32 45 53 55 4D 45 00 9F
ERR	25 52 52 00 C3	ERL	25 52 4C 00 C2
MEM	2D 45 4D 00 C8	FRE	26 52 45 00 DA
RANDOM	32 D2 4F 4D 00 86	STRING$	33 54 52 49 4E 47 24 00 C4
VARPTR	36 41 52 50 54 52 00 C0	POS	30 4F 53 00 DC
FIX	26 49 58 00 F2	CDBL	23 44 42 4C 00 F1
CINT	23 D8 00 EF	CSNG	23 53 4E 47 00 F0
BYE	22 59 45 00 AE		

※ 下划线的子字符串会被V2.0的BASIC输入程序变为代号。

3。主程序。

```
ACB9-  00       NOP             ; 未用。
ACBA-  21C6AC   LD HL,ACC6      ; 【装置记带暂停出口】HL=处理程序首址。
ACBD-  226B7A   LD (7A6B),HL    ; 填入记带程序。
ACC0-  22767A   LD (7A76),HL    ; 同上。
ACC3-  C390B1   JP B190         ; 启动磁带后返回。
ACC6-  3A3B78   LD A,(783B)     ; 【记带暂停处理】取记带前输出锁存器副本。
ACC9-  320068   LD (6800),A     ; 恢复原输出状态。
ACCC-  C3F83A   JP 3AF8         ; 转入V2.0后续程序段。
ACCF-  00       NOP             ; 以下未用。
ACD0-  00       NOP
ACD1-  00       NOP
ACD2-  00       NOP
ACD3-  00       NOP
ACD4-  00       NOP
ACD5-  00       NOP
ACD6-  00       NOP
ACD7-  00       NOP
ACD8-  00       NOP
ACD9-  00       NOP
ACDA-  00       NOP
ACDB-  00       NOP
ACDC-  00       NOP
ACDD-  00       NOP
ACDE-  00       NOP
ACDF-  00       NOP
ACE0-  CD781D   CALL 1D78       ; 【取扩充保留词子程序】取首字符。
ACE3-  280C     JR Z,ACF1       ; 是语句结束符则转去返回V2.0执行驱动程序。
ACE5-  380A     JR C,ACF1       ; 是数字也返回执行驱动程序。
ACE7-  FE7F     CP 7F           ; 测试是不是代号。
ACE9-  D0       RET NC          ; 是代号则返回调用的程序。
ACEA-  CD3D1E   CALL 1E3D       ; 检测字母，是字母返回时NC。
ACED-  3005     JR NC,ACF4      ; 若是字母继续检测下个字符。
ACEF-  C600     ADD A,00        ; 是别的符号则恢复NC状态返回执行驱动程序。
ACF1-  C1       POP BC          ; 清除堆栈中两层调用断点。
```

```
ACF2-  C1       POP BC
ACF3-  C9       RET           ；直接返回执行驱动程序。
ACF4-  23       INC HL        ；指向第二个字符。
ACF5-  CD3D1E   CALL 1E3D     ；测试第二个是否仍是字母。
ACF8-  2B       DEC HL        ；退到首字符。
ACF9-  3013     JR NC,AD0E    ；遇到两个以上字母就去检索扩充保留词表。
ACFB-  FE8E     CP 8E         ；第二个字符是不是RUN代号？
ACFD-  2817     JR Z,AD16     ；是BRUN（DOS命令）则转去置位Cy后返回。
ACFF-  E5       PUSH HL       ；存扫描指针值。
AD00-  21E9B2   LD HL,B2E9    ；HL＝第二字节是代号的保留词表首址。
AD03-  010400   LD BC,0004    ；BC＝表的字节数（共有四个保留词）。
AD06-  EDB1     CPIR          ；检索。
AD08-  E1       POP HL        ；恢复扫描指针值（首字符地址）。
AD09-  2803     JR Z,AD0E     ；检索到相同代号则去检索扩充保留词表。
AD0B-  7E       LD A,(HL)     ；否则将首字符重新取入A。
AD0C-  18E1     JR ACEF       ；转去复位Cy后返回执行驱动程序。
AD0E-  2B       DEC HL        ；扫描指针退到首字符前。
AD0F-  CD5D78   CALL 785D     ；检索扩充保留词表。
AD12-  2002     JR NZ,AD16    ；当前字串不是扩充保留词则去置位Cy后返回。
AD14-  EB       EX DE,HL      ；HL恢复为扫描指针。
AD15-  C9       RET           ；返回调用的程序。
AD16-  37       SCF           ；未找到扩充保留词则将Cy置位。
AD17-  C9       RET           ；返回。
AD18-  C1       POP BC        ；清除堆栈中原存的扫描指针值。
AD19-  B7       OR A          ；按A的内容置Z标志。
AD1A-  C9       RET           ；返回。
AD1B-  D5       PUSH DE       ；【由取元素值进入时的继续】存回入口断点。
AD1C-  D9       EXX           ；恢复入口时各主寄存器内容。
AD1D-  C2781D   JP NZ,1D78    ；若非由此入口进入则取下一字符后返回。
AD20-  E5       PUSH HL       ；存扫描指针值。
AD21-  CDE0AC   CALL ACD0     ；搜索PH扩充保留词。
AD24-  383D     JR C,AD63     ；未找到PH保留词则重新取首字符返回。
AD26-  C1       POP BC        ；清除堆栈中两层调用断点地址。
AD27-  C1       POP BC
```

```
AD28-  C3AE24   JP 24AE        ; 直接返回V2.0取元素值子程序。
AD2B-  D9       EXX            ;【PH语句解释入口程序】存各主寄存器内容。
AD2C-  21A024   LD HL,24A0     ; HL＝由取运算元素值程序调用的断点地址。
AD2F-  D1       POP DE         ; 取出实际的断点地址。
AD30-  B7       OR A           ; 清除Cy，以便作带C标志减法。
AD31-  ED52     SBC HL,DE      ; 比较二者是否相同。
AD33-  28E6     JR Z,AD1B      ; 是求值程序调用则转入取扩充函数保留词。
AD35-  215B1D   LD HL,1D5B     ; HL＝执行驱动程序调用时的断点地址。
AD38-  B7       OR A           ; 清除Cy。
AD39-  ED52     SBC HL,DE      ; 比较二者是否相同。
AD3B-  2806     JR Z,AD43      ; 是执行驱动程序调用则转入取扩充语句保留词。
AD3D-  215020   LD HL,2050     ; HL＝IF工作程序调用时的断点地址。
AD40-  B7       OR A           ; 清除Cy。
AD41-  ED52     SBC HL,DE      ; 比较二者是否相同。
AD43-  D5       PUSH DE        ; 将当前调用断点地址存回原处。
AD44-  D9       EXX            ; 恢复入口处各主寄存器的内容。
AD45-  C2781D   JP NZ,1D78     ; 若是与PH无关的调用则取下一字符返回。
AD48-  E5       PUSH HL        ;【PH扩充语句识别处理】存扫描指针值。
AD49-  CDE0AC   CALL ACE0      ; 取扩充保留词。
AD4C-  3826     JR C,AD74      ; 不是PH扩充保留词则转DOS或V2.0处理。
AD4E-  282E     JR Z,AD7E      ; 是PH保留词则转入解释执行阶段。
AD50-  FE8E     CP 8E          ; 若是代号（NC），是不是RUN?
AD52-  2813     JR Z,AD67      ; 若是RUN,进一步测试是否作为DOS命令。
AD54-  FE82     CP 82          ; 是不是RESET?
AD56-  2808     JR Z,AD60      ; 是则进入中低显模式。
AD58-  FE83     CP 83          ; 是不是SET?
AD5A-  2804     JR Z,AD60      ; 同上。
AD5C-  FE9D     CP 9D          ; 是不是MODE?
AD5E-  20B8     JR NZ,AD18     ; 不是中低显功能的代号则转去返回执行驱动程序。
AD60-  CDBEB1   CALL B1BE      ; 切换到中低显模式。
AD63-  E1       POP HL         ; 恢复保留词前的扫描指针值。
AD64-  C3781D   JP 1D78        ; 转入执行驱动程序。
AD67-  23       INC HL         ;【RUN命令的鉴别】指向代号后字符。
AD68-  7E       LD A,(HL)      ; 取入A。
```

```
AD69-  B7        OR A          ; 测试代码。
AD6A-  28F7      JR A,AD63     ; 若RUN代号后语句结束则是V2.0命令。
AD6C-  FE20      CP 20         ; 是不是空格?
AD6E-  28F7      JR Z,AD67     ; 跳过空格。
AD70-  FE22      CP 22         ; 是不是前引号码?
AD72   20EF      JR NZ,AD63    ; 未跟引号也是V2.0命令（RUN 行号）。
AD74-  CDB4B1    CALL B1B4     ; 4000H空间切换为DOS。
AD77-  E1        POP HL        ; 取回代号前的扫描指针值。
AD78-  ED5B2B78  LD DE,(782B)  ; 将前级系统的RST 10H调用入口地址取出。
AD7C-  D5        PUSH HL       ; 压入堆栈。
AD7D-  C9        RET           ; 返回这个入口重新识别处理当前语句。
AD7E-  C1        POP BC        ;【PH扩充语句执行驱动】清除原存扫描地址。
AD7F-  FE7F      CP 7F         ; 测试代号范围。
AD81-  D0        RET NC        ; 若属V2.0合法代号则返回其执行驱动程序。
AD82-  E3        EX (SP),HL    ; DE=调用断点地址，(栈顶)=扫描指针值。
AD83-  115020    LD DE,2050    ; 令DE=IF工作程序调用断点。
AD86-  B7        OR A          ; 清除Cy。
AD87-  ED52      SBC HL,DE     ; 测试是不是IF语句中的调用。
AD89-  E1        POP HL        ; 恢复扫描指针值。
AD8A-  2804      JR Z,AD90     ; 若是THEN或ELSE后的语句则不再压入断点1D1EH。
AD8C-  111E1D    LD DE,1D1E    ; 是独立语句先设置连锁解释执行的后续入口。
AD8F-  D5        PUSH DE       ; 压入堆栈，以便语句结束返回执行驱动程序。
AD90-  EB        EX DE,HL      ; 扫描指针值存入DE。
AD91-  CDAFB1    CALL B1AF     ; 4000H空间切换为VRAM。
AD94-  2A5379    LD HL,(7953)  ; HL=PH扩充工作子程序相对地址表首址。
AD97-  87        ADD A,A       ; 扩充代号乘以2。
AD98-  4F        LD C,A        ; 存入BC。
AD99-  0600      LD B,00
AD9B-  09        ADD HL,BC     ; HL=代号对应入口在表中的存放地址。
AD9C-  4E        LD C,(HL)     ; 相对地址取入BC。
AD9D-  23        INC HL
AD9E-  46        LD B,(HL)
AD9F-  211AAD    LD HL,AD1A    ; HL=计算入口地址的基址。
ADA2-  09        ADD HL,BC     ; 求出工作子程序的绝对地址。
```

```
ADA3-  E5       PUSH HL         ；入口压入堆栈，以便取下一字符后以RET进入。
ADA4-  C3771D   JP 1D77         ；转去取下个字符后执行工作子程序。
ADA7-  2AFD78   LD HL,(78FD)    ；【PST工作程序】取当前空闲区首址。
ADAA-  E5       PUSH HL         ；存栈。
ADAB-  23       INC HL          ；跳过可能是行号的两字节。
ADAC-  23       INC HL
ADAD-  CD071F   CALL 1F07       ；往下搜索语句结束符00H或：号。
ADB0-  23       INC HL          ；找到后指向下一字节（可能是行指针）。
ADB1-  5E       LD E,(HL)       ；将可能的行指针取入DE。
ADB2-  23       INC HL
ADB3-  56       LD D,(HL)
ADB4-  DF       RST 18          ；单元地址与所存数据比较。
ADB5-  3007     JR NC,ADBE      ；单元地址不小于所存数据肯定不是行指针。
ADB7-  24       INC H           ；将单元地址加256（可能的最大行长）。
ADB8-  DF       RST 18          ；再与所存数据比较。
ADB9-  3805     JR C,ADC0       ；此地址小于所存数据也不是行指针。
ADBB-  EB       EX DE,HL        ；符合条件时暂假定为行指针。
ADBC-  18F3-    JR ADB1         ；沿着指针链继续测试到条件不符时中止。
ADBE-  7A       LD A,D          ；不符合链指针条件则测试是否程序结束标志。
ADBF-  B3       OR E            ；测试DE是否为0000H。
ADC0-  C24A1E   JP NZ,1E4A      ；若不是,则原程序表已破坏，报告“非法调用”。
ADC3-  23       INC HL          ；循指针链到达程序结束标志后指向变量首址。
ADC4-  22F978   LD (78F9),HL    ；恢复为程序终址指针。
ADC7-  E1       POP HL          ；取回扫描开始的地址。
ADC8-  2B       DEC HL          ；后退两字节到首行行指针存贮单元。
ADC9-  2B       DEC HL
ADCA-  74       LD (HL),H       ；置入非零字节以改变程序清除标志。
ADCB-  22A478   LD (78A4),HL    ；HL值作为程序首址指针。
ADCE-  E5       PUSH HL         ；存程序首址。
ADCF-  C3E81A   JP 1AE8         ；转BASIC输入程序，恢复首行行指针。
ADD2-  CAA024   JP Z,24A0       ；【LOMEM工作程序】若缺地址参数则报告出错。
ADD5-  113200   LD DE,0032      ；DE＝字符串区保留字节数。
ADD8-  CD831E   CALL 1E83       ；用CLEAR子程序将串区和堆栈复位。
ADDB-  CD022B   CALL 2B02       ；取用户指定的BASIC程序区首址于DE中。
```

```
ADDE-  21E97A     LD HL,7AE9        ; HL＝程序区常规首址。
ADE1-  ED4BA478   LD BC,(78A4)      ; 取当前程序首址于BC。
ADE5-  7A         LD A.D            ; 测试参数值是否为零。
ADE6-  B3         OR E
ADE7-  200C       JR NZ,ADF5        ; 非零转去按址移动程序块。
ADE9-  E5         PUSH HL           ; LOMEM 0表示恢复到常规首址，将它存栈。
ADEA-  C5         PUSH BC           ; 存当前首址作为传送的程序块源首址。
ADEB-  2B         DFC HL            ; 指向7AE8H单元。
ADEC-  77         LD (HL),A         ; 置入00H供解释程序测试。
ADED-  23         INC HL            ; HL＝目标区首址。
ADEE-  22A478     LD (78A4),HL      ; 作为程序首址指针值。
ADF1-  E5         PUSH HL           ; 再次存栈用作程序块传送的目标首址。
ADF2-  C3D92B     JP 2BD9           ; 用LEVEL II的DELETE功能移动程序块并修改指针。
ADF5-  2B         DEC HL            ; 令HL＝7AE8H。
ADF6-  DF         RST 18            ; 与用户给定的目标首址比较。
ADF7-  D24A1E     JP NC,1E4A        ; 若目标首址≯7AE8H则为非法参数。
ADFA-  2AF978     LD HL,(78F9)      ; 取当前程序终址。
ADFD-  C5         PUSH BC           ; 存当前程序首址。
ADFE-  E5         PUSH HL           ; 存当前程序终址。
ADFF-  ED42       SBC HL,BC         ; HL＝程序的字节数。
AE01-  D5         PUSH DE           ; 存用户给定的目标首址。
AE02-  19         ADD HL,DE         ; HL＝将程序块后移暂存的缓冲区终址。
AE03-  3806       JR C,AE0B         ; 若缓冲区超出FFFFH则报告“内存溢出”。
AE05-  EB         EX DE HL          ; DE＝缓冲区终址。
AE06-  21CEFF     LD HL,FFCE        ; HL＝－50。
AE09-  39         ADD HL,SP         ; HL＝保留50字节堆栈空间后的空闲区终址。
AE0A-  DF         RCT 18            ; 缓冲区是否超出此界？
AE0B-  DA7A19     JP C,197A         ; 若已侵入堆栈也报告“内存溢出”。
AE0E-  D1         POP DE            ; DE＝目标首址。
AE0F-  E1         POP HL            ; HL＝当前程序终址。
AE10-  C1         POP BC            ; BC＝当前程序首址。
AE11-  D5         PUSH DE           ; 再存目标首址。
AE12-  CDDBB0     CALL B0DB         ; 用传送子程序将程序块送入暂存区。
AE15-  E1         POP HL            ; HL＝目标首址。
```

```
AE16-  E5      PUSH HL       ；再存备用。
AE17-  D5      PUSH DE
AE18-  E5      PUSH HL
AE19-  2AD479  LD HL,(79D4)  ；取暂存缓冲区中程序块的终址。
AE1C-  22F978  LD (78F9),HL  ；暂时作为程序区终址供传送使用。
AE1F-  E1      POP HL        ；取出目标首址。
AE20-  AF      XOR A         ；A清零。
AE21-  18C8    JR ADEB       ；转去利用删除程序回传程序块。
AE23-  7E      LD A,(HL)     ；【磁带操作命令#号处理】取当前字符。
AE24-  FE23    CP 23         ；是不是#号?
AE26-  2802    JR Z,AE2A     ；是#号则不改变A的内容（23H）。
AE28-  3E00    LD A,00       ；不是则令A=00H。
AE2A-  324C78  LD (784C),A   ；将23H或00H存入磁带文件提示信息开关单元。
AE2D-  C0      RET NZ        ；非#号则返回。
AE2E-  F7      RST 30        ；否则跳过#号取入下个字符。
AE2F-  C9      RET           ；返回磁带操作命令工作程序。
AE30-  1B      DEC DE        ；【BASIC程序段文件添加程序结束标志】
AE31-  1A      LD A,(DE)     ；从装载终址后退一字节，将代码取入A。
AE32-  B7      OR A          ；是否是零。
AE33-  2808    JR Z,AE3D     ；是零则去核实是否已有结束标志(三个零)。
AE35-  AF      XOR A         ；不是零则只是行中语句结尾符，A清零。
AE36-  12      LD (DE),A     ；由此添上程序结束标志以便修改行指针。
AE37-  13      INC DE
AE38-  12      LD (DE),A
AE39-  13      INC DE
AE3A-  12      LD (DE),A
AE3B-  13      INC DE        ；DE设为程序终址。
AE3C-  C9      RET           ；返回。
AE3D-  1B      DEC DE        ；发现一个零后退一字节。
AE3E-  1A      LD A,(DE)     ；是否还是零?
AE3F-  B7      OR A
AE40-  2803    JR Z,AE45     ；已有两个零继续测试。
AE42-  AF      XOR A         ；否则只是程序行的结尾，A清零。
AE43-  18F2    JR AE37       ；转去在后面添两个零。
```

```
AE45-  1B        DEC DE         ；找到两个零再退一字节。
AE46-  1A        LD A,(DE)      ；是否仍是零？
AE47-  B7        OR A
AE48-  13        INC DE         ；指向下一字节。
AE49-  20EA      JR NZ,AE35     ；若只有两个零转去变为三个零。
AE4B-  13        INC DE         ；是正常程序结束标志则跳过所有的零字节。
AE4C-  13        INC DE         ；指向程序终址。
AE4D-  C9        RET            ；返回。
AE4E-  00        NOP
AE4F-  FE23      CP 23          ；【高显语句#号测试处理】测试#号码。
AE51-  3A4A78    LD A,(784A)    ；取屏幕显示基相码。
AE54-  2006      JR NZ,AE5C     ；无#号则A不变。
AE56-  2F        CPL            ；否则将基相码取反。
AE57-  322A78    LD (782A),A    ；作为图形或汉字显示相位标志。
AE5A-  F7        RST 30         ；取入下一字符。
AE5B-  C9        RET            ；返回工作程序。
AE5C-  2B        DEC HL         ；后退一字节。
AE5D-  18F8      JR AE57        ；转去定义显示相位后重取本字符返回。
AE5F-  78        LD A,B         ；【图形输出控制程序】画点步数存入A。
AE60-  329F78    LD (789F),A    ；作为计数器初值。
AE63-  215C78    LD HL,785C     ；HL＝HPLOT开关地址。
AE66-  36FF      LD (HL),FF     ；打开开关，使PH监控程序绘出已定义图形。
AE68-  7E        LD A,(HL)      ；取开关状态（中断期间绘图完毕后关闭）。
AE69-  B7        OR A           ；测试。
AE6A-  20FC      JR NZ,AE68     ；若未画完则循环等待。
AE6C-  C9        RET            ；画完返回。
AE6D-  3A4778    LD A,(7847)    ；【PH扩充监控程序入口】先测试清屏标志。
AE70-  B7        OR A
AE71-  C262B6    JP NZ,B662     ；非零则转去清除屏幕。
AE74-  3A5C78    LD A,(785C)    ；测试HPLOT开关。
AE77-  B7        OR A
AE78-  CAD7B4    JP Z,B4D7      ；无图形输出则转去测试汉字显示标志。
AE7B-  3A9F78    LD A,(78F9)    ；取绘图步数计数器值。
AE7E-  B7        OR A           ；测试。
```

```
AE7F-   2004       JR NZ,AE85       ；非零则跳转。
AE81-   3ECE       LD A,CE          ；0=256步。每次中断画50点，A=剩余步数。
AE83-   1807       JR AE8C          ；跳转。
AE85-   47         LD B,A           ；待画点数存入B。
AE86-   D632       SUB 32           ；减去本次要画的50点。
AE88-   3806       JR C,AE90        ；若不超过50步就转入绘图。
AE8A-   2804       JR Z,AE90
AE8C-   0632       LD B,32          ；设定画点数为50步。
AE8E-   1804       JR AE94          ；转入绘图。
AE90-   AF         XOR A            ；画最后一次时，A清零。
AE91-   325C78     LD (785C),A      ；关闭HPLOT开关。
AE94-   329F78     LD (789F),A      ；下次待画点数置入计数器。
AE97-   CD9DAE     CALL AE9D        ；调用绘图子程序，画出本次的线段。
AE9A-   C3A7B6     JP B6A7          ；转入V2.0监控程序显示输出之后的部分。
7E9D-   C5         PUSH BC          ；【画线子程序】存需画的步数。
AE9E-   2A5678     LD HL,(7856)     ；HL=从变步长(H—符号，L—纯小数)。
AEA1-   ED5B5878   LD DE,(7858)     ；DE=从变量值(D—整数部分，E—小数部分)。
AEA5-   19         ADD HL,DE        ；求出当前一点的从变量值。
AEA6-   225878     LD (7858),HL     ；存入从变量寄存器。
AEA9-   CB7D       BIT 7,L          ；当前从变量的小数部分是否≥0.1B (0.5)？
AEAB-   2801       JR Z,AEAE        ；不足0.5则舍去。
AEAD-   24         INC H            ；≥0.5则进1。
AEAE-   E5         PUSH HL          ；从变量 (H) 存栈。
AEAF-   3A5478     LD A,(7854)      ；取主变量标志 (X—00H，Y—FFH)。
AEB2-   B7         OR A             ；测试。
AEB3-   215078     LD A,7850        ；HL=HPLOT工作区首址 (X当前值)。
AEB6-   F5         PUSH AF          ；存主变量标志测试状态。
AEB7-   2801       JR Z,AEBA        ；若以X为主变量则跳过下一条指令。
AEB9-   23         INC HL           ；指向Y当前值。
AEBA-   46         LD B,(HL)        ；将主变量当前值取入B。
AEBB-   3A5578     LD A,(7855)      ；主增量符号取入A。
AEBE-   B7         OR A             ；测试。
AEBF-   2802       JR Z,AEC3        ；若为正，则转去加1。
AEC1-   35         DEC (HL)         ；若为负，则减1 (－1－1＋1＝－1)。
```

```
AEC2-  35       DEC (HL)       ; 减1。
AEC3-  34       INC (HL)       ; 加1。
AEC4-  F1       POP AF         ; 取回主变量标志测试状态。
AEC5-  E1       POP HL         ; H＝从变量当前值整数部分。
AEC6-  2003     JR NC,AECB     ; 若是Y为主变量则跳转。
AEC8-  7C       LD A,H         ; 否则将Y送入A（X已在B中）。
AEC9-  1802     JR AECD        ; 跳转。
AECB-  78       LD A,B         ; Y为主变量时将Y值送入A。
AECC-  44       LD B,H         ; X值送入B。
AECD-  212A78   LD HL,782A     ; 取用户定义的图形相位。
AED0-  CB7E     BIT 7,(HL)     ; 测试（00H—与屏幕基相同相，FF—反相）。
AED2-  0EC6     LD C,C6        ; 令C＝SET指令操作码框架。
AED4-  2802     JR Z,AED8      ; 若是正相绘图使用SET指令，跳过下一指令。
AED6-  0E86     LD C,86        ; 反相绘图使用RES指令操作码框架。
AED8-  CD90AF   CALL AF90      ; 以当前X、Y值换算出显示区单元地址(HL)中。
AEDB-  B1       OR C           ; 将字节中的位码（A中）并入操作码框架。
AEDC-  32E0AE   LD (AEE0),A    ; 将操作码填入AEE0H单元。
AEDF-  CBC6     SET 0,(HL)     ; CBH与AEE0H的代码组成SET,RES或BIT指令。
AEE1-  C1       POP BC         ; 取回循环变量（画点数）。
AEE2-  10B9     DJNZ AE9D      ; 减量循环。
AEE4-  C9       RET            ; 画完规定点数返回监控程序。
AEE5-  1809     JR AEF0        ; 【以下9字节是ASCII空格的字符点阵数据】
AEE7-  00       NOP
AEE8-  00       NOP
AEE9-  00       NOP
AEEA-  00       NOP
AEEB-  00       NOP
AEEC-  00       NOP
AEED-  00       NOP
AEEE-  00       NOP
AEEF-  00       NOP
AEF0-  2B       DEC HL         ; 【HPLOT语句解释程序】退到参数之前。
AEF1-  E5       PUSH HL        ; 存扫描指针值。
AEF2-  CD68B4   CALL B468      ; 开中断，以便运行监控程序绘图。
```

```
AEF5-  CDB0B4   CALL B4B0    ；置为高显模式，若由中低显进入则部分清屏。
AEF8-  E1       POP HL       ；取回扫描指针值。
AEF9-  F7       RST 30       ；取参数首字符。
AEFA-  CA38B3   JP Z,B338    ；若无参数则转入语句结束处理。
AEFD-  CD4FAE   CALL AE4F    ；测试处理＃号，返回时已取入下个字符。
AF00-  FEBD     CP BD        ；是不是TO的代号？
AF02-  015278   LD BC,7852   ；BC＝此前的X／Y终值存放地址。
AF05-  280A     JR Z,AF11    ；以TO开头表示以原来终值作为初值。
AF07-  0E50     LD C,50      ；BC指向X／Y初值存放地址（7850H）。
AF09-  CDB0AF   CALL AFB0    ；取X、Y参数对并存入HPLOT工作单元。
AF0C-  FEBD     CP BD        ；测试后面是否是TO的代号。
AF0E-  206D     JR NZ,AF7D   ；未给出终值转去只画一点。
AF10-  00       NOP
AF11-  F7       RST 30       ；取TO后参数首字符。
AF12-  CD4FAE   CALL AE4F    ；测试处理＃号。
AF15-  CDB0AF   CALL AFB0    ；取X、Y参数对并存入工作单元。
AF18-  E5       PUSH HL      ；存扫描指针值。
AF19-  215378   LD HL,7853   ；HL指向Y终值存放单元。
AF1C-  CDC7AF   CALL AFC7    ；求Y方向的步数（E）和符号（D）。
AF1F-  D5       PUSH DE      ；暂存。
AF20-  215278   LD HL,7852   ；HL指向X终值存放单元。
AF23-  CDC7AF   CALL AFC7    ；求X方向的步数和符号（DE）。
AF26-  E1       POP HL       ；取回Y的步数和符号。
AF27-  7B       LD A,E       ；比较X和Y的步数。
AF28-  BD       CP L
AF29-  3E00     LD A,00      ；A清零准备置主变量标志。
AF2B-  015478   LD BC,7854   ；BC为标志单元地址。
AF2E-  F5       PUSH AF      ；存步数比较结果状态(NC—X≥Y,C—X<Y)。
AF2F-  3002     JR NC,AF33   ；若X≥Y则以X为主变量。
AF31-  EB       EX DE,HL     ；否则以Y为主变量，存放的寄存器易位。
AF32-  2F       CPL          ；并使A变为FFH（Y为主变量的标志）。
AF33-  02       LD (BC),A    ；存入主变量标志单元。
AF34-  015178   LD BC,7851   ；BC指向Y初值存放单元。
AF37-  F1       POP AF       ；取步数比较状态。
```

```
AF38-  3001     JR NC,AF3B      ; 若X为主变量则跳过下一指令。
AF3A-  0B       DEC BC          ; Y为主变量时将BC指向X初值。
AF3B-  0A       LD A,(BC)       ; 把从变量初值取入A。
AF3C-  325978   LD (7859),A     ; 作为从变量当前值存放。
AF3F-  D5       PUSH DE         ; 存主增量步数符号。
AF40-  E5       PUSH HL         ; 存从增量步数符号。
AF41-  7A       LD A,D          ; 主增量符号取入A。
AF42-  325578   LD (7855),A     ; 存入主增量符号单元。
AF45-  AF       XOR A           ; A清零。
AF46-  325878   LD (7858),A     ; 清除从变量当前值的小数部分。
AF49-  7D       LD A,L          ; 比较主、从增量步数。
AF4A-  BB       CP E
AF4B-  2007     JR NZ,AF54      ; 若二者不相等则不作以下处理。
AF4D-  E1       POP HL          ; 相等时清除堆栈中的从变量。
AF4E-  2E00     LD L,00         ; L—正方向符号。
AF50-  CBC4     SET 0,H         ; H—步长1。
AF52-  1815     JR AF69         ; 转入后续段。
AF54-  AF       XOR A           ; A清零。
AF55-  65       LD H,L          ; 使HL=从增量绝对值×256。
AF56-  6F       LD L,A
AF57-  57       LD D,A          ; 使DE=主增量。
AF58-  47       LD B,A          ; B清零作为商计数器。
AF59-  ED52     SBC HL,DE       ; 以减代除。
AF5B-  3803     JR C,AF60       ; 余数小于除数时舍去，结束运算。
AF5D-  04       INC B           ; 否则商增1。
AF5E-  18F9     JR AF59         ; 循环减。
AF60-  E1       POP HL          ; 取出从变量步数符号。
AF61-  7C       LD A,H          ; 测试符号。
AF62-  B7       OR A
AF63-  2803     JR Z,AF68       ; 正方向则跳过以下两条指令。
AF65-  AF       XOR A           ; 负方向则将商（B）变为负数的补码。
AF66-  90       SUB B
AF67-  47       LD B,A
AF68-  68       LD L,B          ; 商送入L，H为00H或FFH，HL=B÷256。
```

```
AF69-   22568     LD (7856),HL     ；将带符号的从变步长HL值存入工作区。
AF6C-   C1        POP BC           ；取主变量步数。
AF6D-   41        LD B,C           ；步数作为计数器初值。
AF6E-   04        INC B            ；计数器增1。
AF6F-   CD5FAE    CALL AE5F        ；输出绘图，画完所定义的图形后返回。
AF72-   2A5278    LD HL,(7852)     ；取本次绘图的X、Y终值。
AF75-   225078    LD (7850),HL     ；作为下一次的初值。
AF78-   E1        POP HL           ；恢复扫描指针值。
AF79-   2B        DEC HL           ；后退一字节。
AF7A-   C3F9AE    JP AEF9          ；继续解释执行绘图语句。
AF7D-   E5        PUSH HL          ；【画一点】存扫描指针值。
AF7E-   2178AF    LD HL,AF78       ；预置返回入口地址。
AF81-   E5        PUSH HL          ；压入堆栈以便画完一点后返回此入口。
AF82-   2A5078    LD HL,(7850)     ；取X、Y当前值。
AF85-   225278    LD (7852),HL     ；存为终值。
AF88-   0601      LD B,01          ；计数器设为1次。
AF8A-   C5        PUSH BC          ；存计数器初值。
AF8B-   7C        LD A,H           ；A=Y坐标。
AF8C-   45        LD B,L           ；B=X坐标。
AF8D-   C3CDAE    JP AECD          ；转入画点。
AF90-   C5        PUSH BC          ；【像素地址换算】存X值（列数）。
AF91-   CB3F      SRL A            ；Y值右移，D0→Cy，0→D7。
AF93-   CB18      RR B             ；X值大循环右移，Cy→D7，D0→Cy。
AF95-   CB3F      SRL A            ；循环三次后，由A和B组成的16位寄存器中
AF97-   CB18      RR B               D12～D5等于Y×32，D4～D0等于X÷8的整商。
AF99-   CB3F      SRL A
AF9B-   CB18      RR B
AF9D-   68        LD L,B           ；将B中结果送入L。
AF9E-   67        LD H,A           ；A中结果送入H。HL中是像素字节相对地址。
AF9F-   010040    LD BC,4000       ；令BC=显示区首址。
AFA2-   09        ADD HL,BC        ；加上相对地址，HL=像素所在字节绝对地址。
AFA3-   C1        POP BC           ；取X值。
AFA4-   78        LD A,B           ；送入A。
AFA5-   2F        CPL              ；取列数反码（例如列数001B位数是110B）。
```

```
AFA6-  E607     AND 07       ；屏蔽掉高五位后是像素在单元内的位数。
AFA8-  CB27     SLA A        ；移到位地址码位置（D5～D3）以并入框架。
AFAA-  CB27     SLA A
AFAC-  CB27     SLA A
AFAE-  C9       RET          ；返回（HL是像素的单元地址，A是位码）。
AFAF-  00       NOP
AFB0-  CDBFAF   CALL AFBF    ；【取X、Y参数对子程序】取参数(X)并存贮。
AFB3-  CF       RST 08       ；核实逗号。
AFB4-  2C                    ；逗号的ASCII码2CH。
AFB5-  CDBFAF   CALL AFBF    ；取参数(Y)并存贮。
AFB8-  FEC0     CP C0        ；Y≥192?
AFBA-  D24A1E   JP NC,1E4A   ；若超出最大列数为“非法参数”错误。
AFBD-  7E       LD A,(HL)    ；参数后面的字符取入A。
AFBE-  C9       RET          ；返回。
AFBF-  C5       PUSH BC      ；【取参数并存入工作区】存工作单元地址。
AFC0-  CD1C2B   CALL 2B1C    ；参数（≤255）取入A。
AFC3-  C1       POP BC       ；取回工作单元地址。
AFC4-  02       LD (BC),A    ；存入当前参数。
AFC5-  03       INC BC       ；指向下一个工作单元。
AFC6-  C9       RET          ；返回。
AFC7-  7E       LD A,(HL)    ；【求步数及符号】将终值取入A。
AFC8-  2B       DEC HL       ；指向初值存放单元。
AFC9-  2B       DEC HL
AFCA-  5E       LD E,(HL)    ；取初值入E。
AFCB-  BB       CP E         ；与终值比较。
AFCC-  1600     LD D,00      ；先设D=00H（正号）。
AFCE-  3006     JR NC,AFD6   ；若终值≥初值为正向绘图。
AFD0-  15       DEC D        ；否则令D=FFH（负号）。
AFD1-  47       LD B,A       ；B=终值。
AFD2-  7B       LD A,E       ；A=初值。
AFD3-  90       SUB B        ；初值减终值得到反向绘图步数。
AFD4-  5F       LD E,A       ；步数放入E。
AFD5-  C9       RET          ；返回。
AFD6-  93       SUB E        ；终值减初值得到正向绘图的步数。
```

```
AFD7-  5F        LD E,A        ；存入E。
AFD8-  C9        RET           ；返回。
AFD9-  00        NOP
AFDA-  CD1C2B    CALL 2B1C     ；【计算字库中汉字的地址】取字号。
AFDD-  E5        PUSH HL       ；存扫描指针值。
AFDE-  0605      LD B,05       ；字号乘以32。
AFE0-  CB23      SLA E
AFE2-  CB12      RL D
AFE4-  10FA      DJNZ AFE0     ；DE＝该字号在用户字库中的相对地址。
AFE6-  2AE379    LD HL,(79E3)  ；HL＝字库首址。
AFE9-  19        ADD HL,DE     ；求出绝对地址。
AFEA-  EB        EX DE,HL      ；DE＝字的点阵数据首址。
AFEB-  E1        POP HL        ；恢复扫描指针值。
AFEC-  C9        RET           ；返回。
AFED-  CD23AE    CALL AE23     ；【读磁带文件公共程序】测试命令后的#号。
AFF0-  0620      LD B,20       ；B＝文件类型识别指令填充码。
AFF2-  C5        PUSH BC       ；指令码存栈（C由各工作子程序定义）。
AFF3-  CD8C35    CALL 358C     ；取文件名。
AFF6-  2B        DEC HL        ；扫描指针后退一字节。
AFF7-  F7        RST 30        ；取文件名后字符，测试是否指定装入地址。
AFF8-  3E2F      LD A,2F       ；A＝按记带原址装入方式的填入代码。
AFFA-  2802      JR Z,AFFE     ；文件名后语句结束（未指定地址）则跳转。
AFFC-  3E2B      LD A,2B       ；A＝按指定地址装入方式的填入代码。
AFFE-  3233B0    LD (B033),A   ；填入B033H单元以选择不同的指令块。
B001-  2809      JR Z,B00C     ；无参数则转移。
B003-  CF        RST 08        ；核实逗号。
B004-  2C                      ；逗号码2CH。
B005-  CD022B    CALL 2B02     ；取参数表达式值（即装入的目标首址）。
B008-  ED53D779  LD (79D7),DE  ；存到装入首址单元。
B00C-  C1        POP BC        ；取出文件类型识别指令代码。
B00D-  E5        PUSH HL       ；存扫描指针值。
B00E-  C5        PUSH BC       ；将识别代码存在其上。
B00F-  216436    LD HL,3664    ；HL＝V2.0读磁带文件主程序的首址。
B012-  11277A    LD DE,7A27    ；DE＝存放修改读带程序的缓冲区首址。
```

```
B015-  015700    LD BC,0057      ；传送字节数（截去尾部）。
B018-  EDB0      LDIR            ；将读带程序送入缓冲区。
B01A-  EB        EX DE,HL        ；HL＝缓冲区终址后的地址。
B01B-  36C9      LD (HL),C9      ；在程序末尾放入RET指令码。
B01D-  23        INC HL          ；指向下一单元，将由此装入出错处理子程序。
B01E-  22487A    LD (7A48),HL    ；用此地址取代V2.0读带程序中的原地址。
B021-  224E7A    LD (7A4E),HL
B024-  22737A    LD (7A73),HL
B027-  227C7A    LD (7A7C),HL
B02A-  EB        EX DE,HL        ；将此地址作为目标首址。
B02B-  211137    LD HL,3711      ；HL＝V2.0读带出错处理子程序首址。
B02E-  0E1D      LD C,1D         ；传送字节数。
B030-  EDB0      LDIR            ；将读带出错处理子程序送入新址。
B032-  212FB3    LD HL,B32F      ；用于修改的程序块首址（由前面程序填入）。
B035-  11577A    LD DE,7A57      ；V2.0读带程序中需修改程序段的首址。
B038-  0E08      LD C,08         ；共修改八字节。
B03A-  EDB0      LDIR            ；将指定的程序块送入复盖原址指令。
B03C-  212A7A    LD HL,7A2A      ；HL＝读带出错时新的返回地址。
B03F-  228C7A    LD (7A8C),HL    ；取代出错处理程序中原来的返回地址。
B042-  229A7A    LD (7A9A),HL
B045-  C1        POP BC          ；取回文件类型识别控制码。
B046-  ED43377A  LD (7A37),BC    ；用来修改读带程序的有关指令。
B04A-  CD90B1    CALL B190       ；启动磁带。
B04D-  F3        DI              ；关中断。
B04E-  CD277A    CALL 7A27       ；调用缓冲区内已改造的程序读取磁带文件。
B051-  FB        EI              ；开中断。
B052-  ED53DD79  LD (79DD),DE    ；寄存文件装入终址。
B056-  CD95B1    CALL B195       ；停止磁带。
B059-  E1        POP HL          ；恢复解释程序的扫描指针值。
B05A-  C9        RET             ；返回执行驱动程序。
B05B-  00        NOP
B05C-  CD23AE    CALL AE23       ；【CHAIN工作程序】测试命令后的＃号。
B05F-  CD8C35    CALL 358C       ；取文件名。
B062-  AF        XOR A           ；清除Cy。
```

```
B063-  2AFB78    LD HL,(78FB)     ; HL＝下标变量区首址。
B066-  ED4BF978  LD BC,(78F9)     ; DE＝简单变量区首址。
B06A-  ED42      SBC HL,BC        ; 计算简单变量区字节数。
B06C-  227179    LD (7971),HL     ; 暂存，备恢复时用。
B06F-  ED5BFD78  LD DE,(78FD)     ; DE＝空闲区首址。
B073-  21CEFF    LD HL,FFCE       ; HL＝－50。
B076-  39        ADD HL,SP        ; 保留50字节栈区。
B077-  EB        EX DE,HL         ; HL＝变量暂存缓冲区终址，DE＝变量区终址。
B078-  AF        XOR A            ; 清除Cy表示向高端方向传送。
B079-  CDDBB0    CALL B0DB        ; 将(BC)至(HL)间的全部变量送入暂存区。
B07C-  01F020    LD BC,20F0       ; 定义文件类型识别控制码为T型。
B07F-  2AA478    LD HL,(78A4)     ; 取当前程序首址。
B082-  22D779    LD (79D7),HL     ; 作为指定的装入首址。
B085-  3E2B      LD A,2B          ; 设定指定装入地址方式填充码。
B087-  3233B0    LD (B033),A      ; 修改源程序块传送首址。
B08A-  00        NOP
B08B-  CD0DB0    CALL B00D        ; 读取磁带文件。
B08E-  E5        PUSH HL          ; 装入的程序的首址存栈。
B08F-  CD30AE    CALL AE30        ; 对装入的程序结束标志检查处理。
B092-  D5        PUSH DE          ; 存程序终址。
B093-  ED53F978  LD (78F9),DE     ; 作为新的程序终址指针。
B097-  CDD3B0    CALL B0D3        ; 将暂存区内的原有变量送回当前的变量区。
B09A-  ED53FD78  LD (78FD),DE     ; 传送终址作为新的空闲区指针。
B09E-  2A7179    LD HL,(7971)     ; 取原来的简单变量字节数。
B0A1-  D1        POP DE           ; 取回程序终址。
B0A2-  19        ADD HL,DE        ; 相加得到下标变量区的新首址。
B0A3-  22FB78    LD (78FB),HL     ; 作为指针值。
B0A6-  D1        POP DE           ; 取回程序首址。
B0A7-  D5        PUSH DE          ; 再存。
B0A8-  CDFC1A    CALL 1AFC        ; 修改程序的行指针。
B0AB-  E1        POP HL           ; 取程序首址。
B0AC-  2B        DEC HL           ; 后退一字节。
B0AD-  77        LD (HL),A        ; 将程序前一字节清零。
B0AE-  C9        RET              ; 返回。
```

```
B0AF-   CD23AE     CALL AE23        ；【MERGE工作程序】测试命令后的#号。
B0B2-   CD8C35     CALL 358C        ；取文件名。
B0B5-   2AF978     LD HL,(78F9)     ；取当前程序终址。
B0B8-   2B         DEC HL           ；后退跳过程序结束标志的两个零。
B0B9-   2B         DEC HL           ；HL＝可安放新程序行的地址。
B0BA-   22D779     LD (79D7),HL     ；定为磁带文件装入首址。
B0BD-   3E2B       LD A,2B          ；设定为指定地址装入方式。
B0BF-   3233B0     LD (B033),A      ；填入地址码。
B0C2-   01F020     LD BC,20F0       ；设定文件类型为T型。
B0C5-   CD0DB0     CALL B00D        ；读取磁带文件。
B0C8-   CD30AE     CALL AE30        ；检查处理装入的程序的结束标志。
B0CB-   ED53F978   LD (78F9),DE     ；置入新的程序终址指针。
B0CF-   C3BF36     JP 36BF          ；转去修改行指针后返回等待输入阶段。
B0D2-   00         NOP
B0D3-   ED4BC879   LD BC,(79C8)     ；【通用传送子程序】BC＝暂存区首址。
B0D7-   2AD479     LD HL,(79D4)     ；HL＝暂存区终址。
B0DA-   37         SCF              ；Cy置位表示往低端方向回传。
B0DB-   F5         PUSH AF          ；（送暂存区的入口）存传送方向标志。
B0DC-   C5         PUSH BC          ；存源首址。
B0DD-   3802       JR C,B0E1        ；若是向低端回传则转移。
B0DF-   C1         POP BC           ；取回源首址。
B0E0-   E5         PUSH HL          ；存源终址。
B0E1-   AF         XOR A            ；清除Cy。
B0E2-   ED42       SBC HL,BC        ；求传送字节数。
B0E4-   44         LD B,H           ；字节数放入计数器BC。
B0E5-   4D         LD C,L
B0E6-   E1         POP HL           ；取回源终址。
B0E7-   F1         POP AF           ；取传送方向标志。
B0E8-   03         INC BC           ；计数器增值。
B0E9-   380C       JR C,B0F7        ；若是回传则跳转。
B0EB-   ED53D479   LD (79D4),DE     ；将目标终址记存起来。
B0EF-   EDB8       LDDR             ；从源数据块高端起减址传送。
B0F1-   13         INC DE           ；退到目标首址。
B0F2-   ED53C879   LD (79C8),DE     ；作为暂存区首址。
```

```
B0F6-  C9         RET              ；返回。
B0F7-  EDB0       LDIR             ；回传时从源数据块低端起增址传送。
B0F9-  1B         DEC DE           ；退到目标终址。
B0FA-  ED43C879   LD (79C8),BC     ；将暂存区首址指针清零，表示已空。
B0FE-  C9         RET              ；返回。
B0FF-  0EF0       LD C,F0          ；【SAVET工作程序】设置文件类型为T型。
B101-  AF         XOR A            ；A＝NOP指令码。
B102-  1808       JR B10C          ；转移。
B104-  110EF1     LD DE,F10E       ；B105- LD C,F0【SAVEB工作程序】设B类型码。
B107-  110EFF     LD DE,FF0E       ；B108- LD C,FF【SAVED工作程序】设D类型码。
B10A-  3E13       LD A,13          ；A＝INC DE 指令码（后延一字节为终址）。
B10C-  323FB1     LD (B13F),A      ；填入后边的程序中（否则为空操作）。
B10F-  C5         PUSH BC          ；存类型码。
B110-  CD8C35     CALL 358C        ；取文件名。
B113-  2B         DEC HL           ；退到文件名最后字节。
B114-  F7         RST 30           ；取文件名后面的字符。
B115-  CF         RST 08           ；核实逗号。
B116-  2C                          ；逗号码。
B117-  CD022B     CALL 2B02        ；取用户指定的记带源首址表达式值。
B11A-  D5         PUSH DE          ；存栈。
B11B-  CF         RST 08           ；核实第二个逗号。
B11C-  2C                          ；逗号码。
B11D-  CD022B     CALL 2B02        ；取指定的记带终址。
B120-  E3         EX (SP),HL       ；将扫描指针值存栈，HL＝源首址。
B121-  DF         RST 18           ；比较首、终址。
B122-  D24A1E     JP NC,1E4A       ；若终址不大于首址则为“非法参数”错误。
B125-  E5         PUSH HL          ；存首址。
B126-  D5         PUSH DE          ；存终址。
B127-  21AF34     LD HL,34AF       ；HL＝V2.0写磁带文件程序首址。
B12A-  11307A     LD DE,7A30       ；DE＝存放修改写带程序的缓冲区首址。
B12D-  016200     LD BC,0062       ；传送字节数。
B130-  EDB0       LDIR             ；将写磁带文件程序送到缓冲区。
B132-  21D779     LD HL,79D7       ；HL＝记带首址存放单元地址。
B135-  22447A     LD (7A44),HL     ；用来修改记代程序中的原地址。
```

```
B138-  21DD79    LD HL,79DD      ; HL=记带终址存放单元地址。
B13B-  225A7A    LD (7A5A),HL    ; 用来修改记带程序中的原地址。
B13E-  D1        POP DE          ; 取回记带终址。
B13F-  00        NOP             ;(此指令由不同工作程序填入INC DE或NOP)。
B140-  E1        POP HL          ; 取回记带首址。
B141-  22D779    LD (79D7),HL    ; 存入工作单元供修改后的记带程序使用。
B144-  ED53DD79  LD (79DD),DE
B148-  CDBAAC    CALL ACBA       ; 装置BREAK出口后启动磁带。
B14B-  F3        DI              ; 关中断。
B14C-  110012    LD DE,1200      ; 空循环延时。
B14F-  0600      LD B,00
B151-  10FE      DJNZ B151
B153-  1B        DEC DE
B154-  7A        LD A,D
B155-  B3        OR E
B156-  20F7      JR NZ,B14F
B158-  E1        POP HL          ; 取回扫描指针值。
B159-  C1        POP BC          ; 取回类型码。
B15A-  CD5B35    CALL 355B       ; 写文件名。
B15D-  CD307A    CALL 7A30       ; 用修改过的程序写磁带文件。
B160-  E5        POP HL          ; 存扫描指针值。
B161-  CD95B1    CALL B195       ; 停止磁带。
B164-  E1        POP HL          ; 恢复扫描指针值。
B165-  C9        RET             ; 返回执行驱动程序。
B166-  0EF3      LD C,F3         ;【SAVEW工作程序】设类型码为W型。
B168-  C5        PUSH BC         ; 存类型码。
B169-  CD8C35    CALL 358C       ; 取文件名。
B16C-  2B        DEC DE          ; 后退一字节。
B16D-  F7        RST 30          ; 取文件名后的字符。
B16E-  CF        RST 08          ; 核实逗号。
B16F-  2C                        ; 逗号码。
B170-  CDDAAF    CALL AFDA       ; 求指定字号在用户字库中的地址。
B173-  D5        PUSH DE         ; 存字的数据首址。
B174-  CF        RST 08          ; 核实第二个逗号。
```

```
B175-  2C                          ；逗号码。
B176-  CDDAAF   CALL AFDA          ；求末字在字库中的首地址。
B179-  E5       PUSH HL            ；存扫描指针值。
B17A-  212000   LD HL 0020         ；HL＝一个汉字点阵数据字节数（32）。
B17D-  19       ADD HL,DE          ；由末字首址后移32字节。
B17E-  EB       EX DE,HL           ；DE＝末字数据之后的单元地址。
B17F-  E1       POP HL             ；恢复扫描指针值。
B180-  AF       XOR A              ；令A＝NOP指令码。
B181-  323FB1   LD (B13F),A        ；填入目标单元（以DE为记带终址）。
B184-  189A     JR B120            ；转入记带程序公共段。
B186-  D250B3   JP NC,B350         ；【CALL工作程序】非数字参数则转CALL&程序。
B189-  CD5A1E   CALL 1E5A          ；取目标程序入口地址（直接整型数）。
B18C-  C380B2   JP B280            ；转去作返回安排后转入目标程序入口。
B18F-  00       NOP                ；【以下为系统控制软开关的操作程序】
B190-  21F621   LD HL,21F6         ；【启动磁带】F621- OR 21（将Q5、Q0置1）。
B193-  1809     JR B19E            ；转入软开关操作。
B195-  21E6FE   LD HL,FEE6         ；【停止磁带】E6FE- AND FE（将Q0置0）。
B198-  CD9EB1   CALL B19E          ；调用软开关操作。
B19B-  21F620   LD HL,20F6         ；F620- OR 20（将Q5置1）。
B19E-  F5       PUSH AF            ；【软开关操作】存调用时的A值。
B19F-  3A3B78   LD A,(783B)        ；取输出锁存器副本。
B1A2-  22A5B1   LD (B1A5),HL       ；将操作指令填入目标单元。
B1A5-  E6FD     AND FD             ；执行给定的逻辑运算指令以改变输出数据。
B1A7-  323B78   LD (783B),A        ；将改变了的输出数据存一个副本。
B1AA-  320068   LD (6800),A        ；通过基本I／O接口输出，实现控制。
B1AD-  F1       POP AF             ；恢复原来A的内容。
B1AE-  C9       RET                ；返回。
B1AF-  21F602   LD HL,02F6         ；【4000H空间切换为VRAM】F602- OR 02。
B1B2-  18EA     JR B19E            ；转入操作。
B1B4-  21E6FD   LD HL,FDE6         ；【4000H空间切换为DOS】E6FD- AND FD。
B1B7-  18E5     JR B19E            ；转入操作。
B1B9-  21F6CA   LD HL,CAF6         ；【进入高显模式】F6CA- OR CA。
B1BC-  18E0     JR B19E            ；转入操作。
B1BE-  21E63F   LD HL,3FE6         ；【转入中低显模式】E63F- AND 3F。
```

```
B1C1-  18DB      JR B19E         ；转入操作。
B1C3-  0EF0      LD C.F0         ；【LOADT工作程序】设置文件类型码为T型。
B1C5-  CDEDAF    CALL AFED       ；读入磁带文件。
B1C8-  2AD779    LD HL,(79D7)    ；取文件装入首址。
B1CB-  22A478    LD (78A4),HL    ；作为BASIC程序首址。
B1CE-  2B        DEC HL          ；后退一字节。
B1CF-  3600      LD (HL),00      ；单元清零。
B1D1-  CD30AE    CALL AE30       ；检查处理程序结束标志。
B1D4-  ED53F978  LD (78F9),DE    ；置程序终址指针。
B1D8-  C3BF36    JP 36BF         ；转去修改行指针后返回等待输入阶段。
B1DB-  0EF1      LD C,F1         ；【LOADB工作程序】设置文件类型码为B型。
B1DD-  CDEDAF    CALL AFED       ；读入磁带文件。
B1E0-  7E        LD A,(HL)       ；取命令后的字节。
B1E1-  B7        OR A            ；测试是否行结束符00H。
B1E2-  C0        RET NZ          ；若非零（可能是：号）继续执行原程序命令。
B1E3-  E5        PUSH HL         ；若是行的结束存扫描指针值。
B1E4-  2AA278    LD HL,(78A2)    ；取当前行号。
B1E7-  23        INC HL          ；行号加1。
B1E8-  7C        LD A,H          ；测试当前行号是否为FFFFH（直接命令）。
B1E9-  B5        OR L
B1EA-  E1        POP HL          ；恢复扫描指针值。
B1EB-  C0        RET NZ          ；若是程序语句则继续往下解释执行。
B1EC-  C3191A    JP 1A19         ；若是直接命令的结束则转入等待输入阶段。
B1EF-  0EFF      LD C,FF         ；【LOADD工作程序】设置文件类型码为D型。
B1F1-  18EA      JR B1DD         ；其余与LOADT相同。
B1F3-  3E21      LD A,21         ；【LOADW工作程序】A中为LD HL,NN操作码。
B1F5-  32AAB2    LD (B2AA),A     ；填入恢复堆栈子程序，将它打开以备执行。
B1F8-  C1        POP BC          ；取出堆栈中最近一个返回地址存于BC。
B1F9-  EB        EX DE,HL        ；程序扫描指针存入DE。
B1FA-  210000    LD HL,0000      ；HL清零。
B1FD-  39        ADD HL,SP       ；将当前堆栈指针值取入HL。
B1FE-  22C579    LD (79C5),HL    ；暂存起来。
B201-  31FFB7    LD SP,B7FF      ；将堆栈迁到内存高端，以免与装入字库冲突。
B204-  C5        PUSH BC         ；存入原返回地址。
```

```
B205-  EB        EX DE,HL        ；取回扫描指针。
B206-  0EF3      LD C,F3         ；设置文件类型为W型。
B208-  CDEDAF    CALL AFED       ；读入磁带文件。
B20B-  E5        PUSH HL         ；存扫描指针值。
B20C-  00        NOP
B20D-  2AD779    LD HL,(79D7)    ；HL＝文件装入首址。
B210-  ED5BE379  LD DE,(79E3)    ；DE＝原字库首址。
B214-  DF        RST 18          ；二者比较。
B215-  380B      JR C,B222       ；若新字库低端已超出原字库则转移。
B217-  00        NOP
B218-  D1        POP DE          ；未超出原字库范围时取扫描指针值于DE。
B219-  C1        POP BC          ；取出原返回地址。
B21A-  2AC579    LD HL,(79C5)    ；HL＝原堆栈指针值（此时原堆栈未破坏）。
B21D-  F9        LD SP HL        ；还原堆栈指针。
B21E-  C5        PUSH BC         ；存回返回地址。
B21F-  EB        EX DE,HL        ；恢复扫描指针。
B220-  18BE      JR B1E0         ；转去处理语句的结束，继续执行程序。
B222-  00        NOP             ；【装入的字库已复盖原来堆栈时的处理】
B223-  E5        PUSH HL         ；文件装入首址存临时堆栈。
B224-  22E379    LD (79E3),HL    ；作为新的字库指针。
B227-  AF        XOR A           ；清除Cy。
B228-  EB        EX DE,HL        ；HL＝原首址，DE＝新首址。
B229-  ED52      SBC HL,DE       ；求出增加的字节数。
B22B-  0605      LD B,05         ；除以32得到增加的汉字数。
B22D-  CB3C      SRL H
B22F-  CB1D      RR L
B231-  10FA      DJNZ B22D
B233-  00        NOP
B234-  3AE279    LD A,(79E2)     ；取原字数。
B237-  85        ADD A,L         ；加上新增字数。
B238-  32E279    LD (79E2),A     ；修改汉字库字数指针单元内容。
B23B-  2AB178    LD HL,(78B1)    ；HL＝原内存终端地址。
B23E-  ED5BA078  LD DE,(78A0)    ；DE＝原字符串区首址。
B242-  AF        XOR A           ；清除Cy。
```

```
B243-  ED52      SBC HL,DE       ；求出原定义的字符串区字节数。
B245-  EB        EX DE,HL        ；字节数放入DE以便重辟串区。
B246-  E1        POP HL          ；取回文件装入首址。
B247-  2B        DEC HL          ；后退一字节。
B248-  22B178    LD (78B1),HL    ；作为新的内存终端指针值。
B24B-  C3841E    JP 1E84         ；转入CLEAR程序初始化串区、堆栈和变量区。
B24E-  00        NOP
B24F-  00        NOP
B250-  110000    LD DE,0000      ；【取十六进制直接数程序】设DE初值为0。
B253-  CDABB6    CALL B6AB        取首数码，合法数码在A中，NC。
B256-  DA9719    JP C,1997       ；若不是十六进制数码则为“语法错误”。
B259-  0E04      LD C,04         ；以C为数码计数器，最多允许四位数。
B25B-  83        ADD A,F         ；将本位值加入DE。
B25C-  5F        LD E,A
B25D-  0D        DEC C           ；计数减1。
B25E-  280F      JR Z,B26F       ；若已取完四位数码则转去结束。
B260-  CDABB6    CALL B6AB       ；取下个数码。
B263-  380B      JR C,B270       ；若不再是十六进制数码则转去结束。
B265-  0604      LD B,04         ；将以前各位的值乘以16（加权）。
B267-  CB23      CLA E
B269-  CB12      RL D
B26B-  10FA      DJNZ B267
B26D-  18EC      JR B25B         ；再加本位值后取下一数码。
B26F-  F7        RST 30          ；取入数后字符。
B270-  EB        EX DE,HL        ；HL＝十六进制数值，DE＝扫描指针值。
B271-  CD9A0A    CALL 0A9A       ；将HL值放入WRA1，并给出整型标志。
B274-  EB        EX DE,HL        ；恢复扫描指针值。
B275-  C9        RET             ；返回。
B276-  00        NOP
B277-  0EF1      LD C,F1         ；【CALLB工作程序】设置文件类型为B型。
B279-  CDEDAF    CALL AFED       ；读入磁带文件。
B27C-  ED5BD779  LD DE,(79D7)    ；将装入首址取入DE。
B280-  E5        PUSH HL         ；存当前扫描指针值供返回继续执行BASIC程序。
B281-  219008    LD HL,0890      ；设返回地址（0890H有POP HL，RET二指令）。
```

```
B284-  E5      PUSH HL        ；存栈。
B285-  EB      EX DE,HL       ；HL＝装入的机器语言程序首址。
B286-  E9      JP (HL)        ；转入执行（遇RET返回到0890H）。
B287-  C1      POP BC         ；【CALLT工作程序】取出返回地址（1D1EH）。
B288-  E5      PUSH HL        ；存入当前扫描指针值。
B289-  E5      PUSH HL        ；再存一组。
B28A-  2AA278  LD HL,(78A2)   ；取当前行号。
B28D-  E3      EX (SP),HL     ；行号存入堆栈，HL＝扫描指针。
B28E-  3E91    LD A,91        ；A装入GOSUB代号。
B290-  F5      PUSH AF        ；存栈供调用的磁带子程序用RETURN返回主程序。
B291-  33      INC SP         ；堆栈后退一字节，排出同A存入的F的内容。
B292-  C5      PUSH BC        ；压入执行驱动程序入口，形成GOSUB数据组。
B293-  0EF0    LD C,F0        ；设置文件类型为T型。
B295-  CDEDAF  CALL AFED      ；读入磁带文件。
B298-  CD30AE  CALL AE30      ；检查处理程序结束标志。
B29B-  2AD779  LD HL,(79D7)   ；取装入首址。
B29E-  E5      PUSH HL        ；存栈。
B29F-  CDFB1A  CALL 1AFB      ；修改行指针。
B2A2-  E1      POP HL         ；取回程序首址。
B2A3-  2B      DEC HL         ；后退一字节作为扫描指针初值。
B2A4-  3600    LD (HL),00     ；将此字节清零。
B2A6-  C9      RET            ；返回执行驱动程序，执行装入的BASIC子程序。
B2A7-  CD95B1  CALL B195      ；【恢复堆栈程序】停止磁带(以下为LOADW用)。
B2AA-  210000  LD HL,0000     ；将HL清零（非LOADW调用时此处为RET指令）。
B2AD-  39      ADD HL,SP      ；HL＝堆栈指针值。
B2AE-  1178AB  LD DE,AB78     ；DE＝PH系统堆栈底下一单元地址。
B2B1-  DF      RST 18         ；二者比较。
B2B2-  D8      RET C          ；若是正常堆栈则返回，不作处理。
B2B3-  C1      POP BC         ；否则是内存终端的临时堆栈，取出返回地址。
B2B4-  2AC579  LD HL,(79C5)   ；取LOADW工作程序寄存的原堆栈指针值。
B2B7-  F9      LD SP,HL       ；堆栈到恢复到原来位置。
B2B8-  C5      PUSH BC        ；存入返回地址。
B2B9-  3EC9    LD A,C9        ；将B2AAH单元改为RET指令，关闭入口，使得非
B2BB-  32AAB2  LD (B2AA),A      LOADW工作过程中都不执行此程序段。
```

```
B2BE-  C9       RET             ；返回。
B2BF-  00       NOP
B2C0-  E5       PUSH HL         ；【HGR工作程序】存扫描指针值。
B2C1-  FE23     CP 23           ；测试＃号。
B2C3-  3E03     LD A,03         ；先设全屏清除控制码。
B2C5-  324778   LD (7847),A     ；存入清屏控制单元。
B2C8-  F5       PUSH AF         ；存＃号测试结果状态（有＃号Z置位）。
B2C9-  CD65B4   CALL B465       ；进入高显状态。
B2CC-  214A78   LD HL,784A      ；HL＝屏幕基相标志地址。
B2CF-  AF       XOR A           ；A清零。
B2D0-  77       LD (HL),A       ；预置为正相。
B2D1-  F1       POP AF          ；取回＃号测试状态。
B2D2-  E5       PUSH HL         ；存基相标志地址。
B2D3-  210000   LD HL,0000      ；HL清零。
B2D6-  22B079   LD (79B0),HL    ；汉字光标（行列地址指针）复位为0行0列。
B2D9-  225078   LD (7850),HL    ；绘图原点复位至屏幕左上角（X＝0，Y＝0）。
B2DC-  E1       POP HL          ；取回基相标志地址。
B2DD-  2006     JR NZ,B2E5      ；若无＃号则保持正相不变。
B2DF-  35       DEC (HL)        ；有＃号将基相标志变为FFH（反相）。
B2E0-  E1       POP HL          ；恢复扫描指针值。
B2E1-  F7       RST 30          ；取下个字符。
B2E2-  C338B3   JP B338         ；转入高显语句结束处理。
B2E5-  E1       POP HL          ；后无＃号则恢复扫描指针值。
B2E6-  C338B3   JP B338         ；转入高显语句结束处理。
B2E9-  B2       OR D            ；【PH扩充保留词所含代号表】HPRINT(B2)。
B2EA-  C6D2     ADD A,D2        ；HPOINT（C6），RANDOM（D2）。
B2EC-  D8       RET C           ；CINT（D8）。
B2ED-  F7       RST 30          ；【HPOINT函数子程序】取下一字符。
B2EE-  CF       RST 08          ；核实左括号。B2EF- 28  左括号码。
B2EF-  28E5                     ；B2F0- E5  PUSH HL  存扫描指针值。
B2F1-  CDAFB1   CALL B1CF       ；4000H空间切换为VRAM。
B2F4-  E1       POP HL          ；取回扫描指针值。
B2F5-  015A78   LD BC,785A      ；BC＝HPOINT寄存器地址。
B2F8-  CDB0AF   CALL AFB0       ；取X、Y参数对存入HPOINT寄存器。
```

```
B2FB-  E5        PUSH HL          ；存扫描指针值。
B2FC-  110DB3    LD DE,B30D       ；设置继续执行的返回地址。
B2FF-  D5        PUSH DE          ；压入堆栈以便用RET转入。
B300-  0601      LD B,01          ；设循环变量为1（测一点）。
B302-  C5        PUSH BC          ；存循环变量。
B303-  2A5A78    LD HL,(785A)     ；取X、Y值。
B306-  7C        LD A,H           ；A=Y。
B307-  45        LD B,L           ；B=X。
B308-  0E46      LD C,46          ；C=BIT指令操作码框架。
B30A-  C3D8AE    JP AED8          ；转入绘、抹、测点程序，完后返回B30DH。
B30D-  210000    LD HL,0000       ；【测点结束的返回入口】HL预置为零。
B310-  2801      JR Z,B313        ；若目标点测试结果为零则函数值为0。
B312-  23        INC HL           ；否则HL=1。
B313-  CD9A0A    CALL 0A9A        ；HL内容作为函数值存入WRA1。
B316-  00        NOP
B317-  00        NOP
B318-  00        NOP
B319-  E1        POP HL           ；恢复扫描指针值。
B31A-  C32F25    JP 252F          ；转去核实右括号后返回表达式求值程序。
B31D-  F5        PUSH AF          ；【清除汉字显示标志并等待清完屏幕】存A值。
B31E-  AF        XOR A            ；A清零。
B31F-  32AF79    LD (79AF),       ；清除汉字显示标志。
B322-  3A4778    LD A,(7847)      ；取清屏标志。
B325-  B7        OR A             ；测试。
B326-  20FA      JR NZ,B322       ；未完成清屏则循环等待。
B328-  F1        POP AF           ；取回A内容。
B329-  C9        RET              ；返回。
B32A-  00        NOP
B32B-  ED5BD779  LD DE,(79D7)     ；【修改读带程序的源程序块】DE=装入首址。
B32F-  ED53D779  LD (79D7),DE     ；将DE作为装入首址。
B333-  00        NOP              ；以下4条NOP指令，可使传送8字节既能从B32BH
B334-  00        NOP                开始，也能从B32FH开始。前者实现以用户给
B335-  00        NOP                定的地址（79D7H中）为装入首址，后者则仍
B336-  00        NOP                以原来程序的记带首址（DE）为装入首址。
```

```
B337-  00       NOP
B338-  CD1DB3   CALL B31D        ；【高显语句结束处理】清除汉字标志。
B33B-  AF       XOR A            ；A清零。
B33C-  BE       CP (HL)          ；测试当前字符是不是行结束符。
B33D-  C0       RET NZ           ；字符后有语句则返回执行驱动程序继续执行。
B33E-  E5       PUSH HL          ；存扫描指针值。
B33F-  2AA278   LD HL,(78A2)     ；取当前行号。
B342-  23       INC HL           ；行号加1。
B343-  7C       LD A,H           ；测试当前是否是执行直接命令（行号FFFFH）。
B344-  B5       OR L
B345-  2007     JR NZ,B34E       ；若在程序中则返回执行驱动程序解释执行下行。
B347-  CDF42E   CALL 2EF4        ；否则扫描键盘一次。
B34A-  FE01     CP 01            ；测试是否按了BREAK键。
B34C-  20F9     JR NZ,B347       ；未按则保持高显画面继续扫描键盘。
B34E-  E1       POP HL           ；按BREAK键后，恢复程序扫描指针值。
B34F-  C9       RET              ；返回执行驱动程序（因命令结束回到输入阶段）。
B350-  CF       RST 08           ；【CALL&程序段】核实&号，B351- 26  &号码。
B351-  262B                      ；B352- 2B  DEC HL  后退到&号。
B353-  CD9479   CALL 7994        ；取十六进制地址参数。
B356-  C380B2   JP B280          ；转入CALLB程序后段。
B359-  E5       PUSH HL          ；【HPRINT解释程序】存扫描指针值。
B35A-  F5       PUSH AF          ；存保留词后跟字符及测试状态。
B35B-  CD68B5   CALL B468        ；开中断并打开中断服务程序出口以显示汉字。
B35E-  CD70B4   CALL B470        ；等待原有显示信息输出完毕。
B361-  AF       XOR A            ；A清零。
B362-  324878   LD (7848),A      ；预置为正相输入。
B365-  CDB0B4   CALL B4B0        ；根据原来的显示模式部分清屏。
B368-  3EFF     LD A,FF          ；置汉字显示标志值为FFH。
B36A-  32AF79   LD (79AF),A
B36D-  F1       POP AF           ；取回保留词后字符及测试状态。
B36E-  E1       POP HL           ；恢复扫描指针值。
B36F-  286F     JR Z,B3E0        ；若只是“HPRINT”则转去换行并结束本语句。
B371-  FE40     CP 40            ；测试是否跟有@号（定位显示符）。
B373-  201C     JR NZ,B391       ；不是@号则转移。
```

```
B375-  F7        RST 30          ; 取@号后字符。
B376-  CF        RST 08          ; 核实左括号。B377- 28  左括号码。
B377-  28CD                      ; B378- CD1C2B  CALL 2B1C  取行位置参数。
B379-  1C
B37A-  2B
B37B-  D5        PUSH DE         ; 行数在E和A中，存栈。
B37C-  CF        RST 08          ; 核实逗号。
B37D-  2C                        ; 逗号码。
B37E-  CD1C2B    CALL 2B1C       ; 取列位置参数，在E和A中。
B381-  D1        POP DE          ; E=行数。
B382-  57        LD D,A          ; D=列数。
B383-  CF        RST 08          ; 核实右括号。
B384-  29                        ; 右括号码。
B385-  CF        RST 08          ; 核实；号。
B386-  3B                        ; ；号码。
B387-  ED53B079  LD (79B0),DE    ; 将用户给出的行列位置置为指针值。
B38B-  284B      JR Z,B3D8       ; 若语句结束则只改变汉字光标，转结束处理。
B38D-  2B        DEC HL          ; 退一字节（此三条指令在再次转入时用）。
B38E-  F7        RST 30          ; 取下一字符。
B38F-  2875      JR Z,B406       ; 若语句结束则换行后转入结束处理。
B391-  FE23      CP 23           ; 是不是#号?
B393-  2008      JR NZ,B39D      ; 不是#号则跳转。
B395-  3E01      LD A,01         ; 设置输出反相字符标志。
B397-  324878    LD (7848),A
B39A-  F7        RST 30          ; 取下个字符。
B39B-  2869      JR Z,B406       ; 若语句结束则换行后转入结束处理。
B39D-  FE5B      CP 5B           ; 是否［号（汉字码标识）？
B39F-  2049      JR NZ,B3EA      ; 非汉字码转移。
B3A1-  F7        RST 30          ; 指向下一字符。
B3A2-  CD1C2B    CALL 2B1C       ; 取字号。
B3A5-  3AE279    LD A,(79E2)     ; 测试是否超出当前字库最大字号。
B3A8-  BB        CP E
B3A9-  3815      JR C,B3C0       ; 若超过为"非法参数"错误。
B3AB-  7E        LD A,(HL)       ; 将字号后字符取入A。
```

```
B3AC-  FE5C     CP 5C          ；是＼号（词组标识）吗?
B3AE-  3E00     LD A,00        ；A清零。
B3B0-  2011     JR NZ,B3C3     ；若不是＼号（单字）则转移。
B3B2-  D5       PUSH DE        ；存首字号。
B3B3-  F7       RST 30         ；指向下一字符。
B3B4-  CD1C2B   CALL 2B1C      ；取末字号。
B3B7-  3AE279   LD A,(79E2)    ；与最大字号比较。
B3BA-  BB       CP E
B3BB-  3803     JR C,B3C0      ；若超出允许字号为“非法调用”错误。
B3BD-  7B       LD A,E         ；末字号存入A。
B3BE-  D1       POP DE         ；取回首字号（E中）。
B3BF-  93       SUB E          ；求字号差。
B3C0-  DA4A1E   JP C,1E4A      ；若末字号小于首字号为“非法调用”错误。
B3C3-  1815     JR B3DA        ；转到后续程序段。
B3C5-  E5       PUSH HL        ；【显示码输出子程序】存扫描指针值。
B3C6-  47       LD B,A         ；字号差放入B。
B3C7-  04       INC B          ；加1即为词组的字数，作为循环变量。
B3C8-  CD76B4   CALL B476      ；取输出缓冲区指针(HL)和当前输出相位码(A)。
B3CB-  F3       DI             ；关中断。
B3CC-  77       LD (HL),A      ；将相位码送入输出缓冲区。
B3CD-  23       INC HL         ；指向输出缓冲区下一单元。
B3CE-  73       LD (HL),E      ；字号送入下一单元。
B3CF-  00       NOP
B3D0-  CD80B4   CALL B480      ；输出显示。
B3D3-  1C       INC E          ；E=下一字号。
B3D4-  10F2     DJNZ B3C8      ；词组的字尚未输出完毕则循环输出。
B3D6-  E1       POP HL         ；恢复扫描指针值。
B3D7-  C9       RET            ；返回。
B3D8-  183B     JR B415        ；【由B38BH来的跳转接力地址】
B3DA-  CDC5B3   CALL B3C5      ；【HPRINT解释程序（续）】显示码输出显示。
B3DD-  00       NOP
B3DE-  2B       DEC HL         ；退到字号的末尾。
B3DF-  F7       RST 30         ；取下一字符。
B3E0-  2824     JR Z,B406      ；若语句结束则换行转入结束处理。
```

```
B3E2-  FE2C     CP 2C         ；是不是逗号（汉字分隔符）？
B3E4-  28BB     JR Z,B3A1     ；是逗号转去取下一字号。
B3E6-  CF       RST 08        ；否则核实 ] 号。
B3E7-  5D                     ；] 号码。
B3E8-  281C     JR Z,B406     ；] 号后语句结束则换行转入结束处理。
B3EA-  FE2C     CP 2C         ；是否逗号（分段显示）？
B3EC-  2010     JR NZ,B3FE    ；不是则测试；号。
B3EE-  E5       PUSH HL       ；存扫描指针值。
B3EF-  1E0F     LD E,0F       ；E＝分段显示控制码。
B3F1-  CD76B4   CALL B476     ；取输出缓冲区指针。
B3F4-  F3       DI            ；关中断。
B3F5-  CD9AB4   CALL B49A     ；输出显示。
B3F8-  E1       POP HL        ；恢复扫描指针值。
B3F9-  F7       RST 30        ；取下一字符。
B3FA-  188F     JR B38B       ；继续扫描语句剩余部分。
B3FC-  00       NOP
B3FD-  00       NOP
B3FE-  FE3B     CP 3B         ；测试；号。
B400-  201D     JR NZ,B41F    ；不是；号就去取表达式值。
B402-  18F5     JR B3F9       ；是；号不作处理，继续扫描语句剩余部分。
B404-  1887     JR B38D       ；【来自B431H，B446H的长距跳转接力地址】
B406-  F5       PUSH AF       ；【汉字换行】存A的内容及标志状态。
B407-  E5       PUSH HL       ；存扫描指针值。
B408-  AF       XOR A         ；A清零。
B409-  324878   LD (7848),A   ；清除汉字显示标志。
B40C-  1E0D     LD E,0D       ；E＝汉字换行控制码。
B40E-  CD91B4   CALL B491     ；输出换行。
B411-  E1       POP HL        ；恢复扫描指针值。
B412-  F1       POP AF        ；恢复A的内容和标志状态。
B413-  1805     JR B41A       ；转入结束处理。
B415-  E5       PUSH HL       ；【等待输出完毕】存扫描指针值。
B416-  CD70B4   CALL B470     ；等待显示信息输出完毕并取缓冲区指针。
B419-  E1       POP HL        ；恢复扫描指针值。
B41A-  C338B3   JP B338       ；转入结束处理。
```

```
B41D-  00        NOP
B41E-  00        NOP
B41F-  CD3723    CALL 2337      ；求参数表达式的值。
B422-  E5        PUSH HL        ；存扫描指针值。
B423-  E7        RST 20         ；测试数据类型。
B424-  2815      JR Z,B43B      ；若是字符串则转移。
B426-  CDBD0F    CALL 0FBD      ；将数值变成ASCII码串，返回时HL是存放代码
B429-  00        NOP              串的缓冲区首址。
B42A-  00        NOP
B42B-  00        NOP
B42C-  7E        LD A,(HL)      ；从缓冲区取一字符码。
B42D-  B7        OR A           ；测试是否结束符00H。
B42E-  2003      JR NZ,B433     ；不是则转去输出此字符，然后取下一字符码。
B430-  E1        POP HL         ；ASCII码串结束则恢复扫描指针值。
B431-  18D1      JR B404        ；转去扫描语句的剩余部分。
B433-  E5        PUSH HL        ；存缓冲区地址。
B434-  CD90B4    CALL B490      ；输出显示。
B437-  E1        POP HL         ；取回缓冲区指针。
B438-  23        INC HL         ；指向下一单元。
B439-  18F1      JR B42C        ；继续输出显示ASCII字符直至代码串结束。
B43B-  CDDA29    CALL 29DA      ；取字符串地址（存入BC）。
B43E-  CDC409    CALL 09C4      ；取字符串的长度（存入D）。
B441-  14        INC D          ；测试D是否为0（空字符串）。
B442-  15        DEC D
B443-  2003      JR NZ,B448     ；若不是空串则转去输出显示。
B445-  E1        POP HL         ；否则恢复扫描指针值。
B446-  18BC      JR B404        ；继续扫描语句剩余部分。
B448-  0A        LD A,(BC)      ；取一个待显示字符码。
B449-  FEDB      CP DB          ；是否大于反白的“Z”？
B44B-  3014      JR NC,B461     ；大于DBH的代码转去显示一个空格。
B44D-  FEC1      CP C1          ；是否小于反白的“A”？
B44F-  380C      JR C,B45D      ；小于C1H则继续测试。
B451-  D660      SUB 60         ；反白字母的代码减去60H变为小写字母代码。
B453-  D5        PUSH DE        ；存字符数。
```

```
B454-  C5      PUSH BC        ；存当前缓冲区地址。
B455-  CD90B4  CALL B490      ；输出显示。
B458-  C1      POP BC         ；取回缓冲区当前地址。
B459-  D1      POP DE         ；取回字数。
B45A-  03      INC BC         ；指向下一个字符码。
B45B-  18E5    JR B442        ；继续输出。
B45D-  FE81    CP 81          ；代码是否小于81H?
B45F-  38F2    JR C,B453      ；小于81H为正常字符显示。
B461-  3E20    LD A,20        ；否则变为空格码。
B463-  18EE    JR B453        ；输出显示一个空格。
B465-  CDB9B1  CALL B1B9      ；【打开高分辨率显示程序入口】进入高显。
B468-  217D78  LD HL,787D     ；打开中断服务程序出口。
B46B-  36C3    LD (HL),C3
B46D-  FB      EI             ；开中断。
B46E-  C9      RET            ；返回。
B46F-  00      NOP
B470-  3AAF7A  LD A,(7AAF)    ；【等待输出缓冲区空】取缓冲区计数器值。
B473-  B7      OR A           ；测试。
B474-  20FA    JR NZ,B470     ；原有内容未输出完则循环等待。
B476-  2AB07A  LD HL,(7AB0)   ；缓冲区空后取当前指针。
B479-  3A4878  LD A,(7848)    ；取当前输出相标志。
B47C-  C9      RET            ；返回。
B47D-  00      NOP
B47E-  00      NOP
B47F-  00      NOP
B480-  23      INC HL         ；【修改输出缓冲区指针】缓冲区地址增1。
B481-  22B07A  LD (7AB0),HL   ；作为新的指针值。
B484-  21AF7A  LD HL,7AAF     ；HL＝缓冲区计数器地址。
B487-  34      INC (HL)       ；计数器增1。
B488-  00      NOP
B489-  FB      EI             ；开中断。
B48A-  3E01    LD A,01        ；设置输出汉字控制数(每次中断输出一个字)。
B48C-  BE      CP (HL)        ；与缓冲区计数值比较。
B48D-  38FB    JR C,B48A      ；若已有二个字符代码暂停送入新字。
```

```
B48F-  C9       RET           ；返回送入下一个字。
B490-  5F       LD E,A        ；【显示码送入缓冲区】字符码存入E。
B491-  CD76B4   CALL B476     ；取缓冲区指针和输出相位标志。
B494-  F3       DI            ；关中断。
B495-  B7       OR A          ；测试当前输出相位。
B496-  2802     JR Z,B49A     ；若是正相就将显示码直接送入缓冲区。
B498-  CBFB     SET 7,E       ；是反相输出时将字符码D7置1。
B49A-  73       LD (HL),E     ；显示码送入缓冲区当前单元。
B49B-  18E3     JR B480       ；去修改缓冲区指针后继续进行。
B49D-  00       NOP
B49E-  00       NOP
B49F-  00       NOP
B4A0-  3E20     LD A,20       ；【行末消空】A＝行长（32列）。
B4A2-  94       SUB H         ；减去已显示列数。
B4A3-  47       LD B,A        ；以应补的空格数为循环变量。
B4A4-  C5       PUSH BC       ；存变量值。
B4A5-  3E20     LD A,20       ；A＝ASCII空格码。
B4A7-  CD90B4   CALL B490     ；送入显示缓冲区。
B4AA-  C1       POP BC        ；取回循环变量。
B4AB-  10F7     DJNZ B4A4     ；循环输出所需空格码。
B4AD-  C9       RET           ；返回。
B4AE-  00       NOP
B4AF-  00       NOP
B4B0-  3A3B78   LD A,(783B)   ；【进入高显时部分清屏】取输出锁存器副本。
B4B3-  E6C8     AND C8        ；保留Q7、Q6、Q3，其余位清零。
B4B5-  FEC8     CP C8         ；以上三位是否都为1（原在高显状态，Z）。
B4B7-  214778   LD HL,7847    ；HL＝清屏标志单元地址。
B4BA-  3600     LD (HL),00    ；清屏代码预置为零（不清屏）。
B4BC-  F5       PUSH AF       ；存原显示模式测试状态。
B4BD-  3A3B78   LD A,(783B)   ；再取输出锁存器副本。
B4C0-  F6CA     OR CA         ；置为高显模式输出码。
B4C2-  323B78   LD (783B),A   ；存副本。
B4C5-  320068   LD (6800),A   ；输出。
B4C8-  F1       POP AF        ；取原有模式测试状态。
```

```
B4C9-  C8        RET Z          ；若原已在高显中则返回，不必部分清屏。
B4CA-  34        INC (HL)       ；否则将清屏标志置为1（清除7000H～71FFH）。
B4CB-  CB5F      BIT 3,A        ；测试Q3。
B4CD-  C8        RET Z          ；若Q3为0（低显模式）则返回。
B4CE-  34        INC (HL)       ；由MODE(1)转入时为02H（清除7000H～77FFH）。
B4CF-  C9        RET            ；返回，在下次中断时按标志执行清屏操作。
B4D0-  3A4778    LD A,(7847)    ；【PH监控程序入口之一】取清屏标志。
B4D3-  B7        OR A           ；测试。
B4D4-  C265B6    JP NZ,B665     ；若标志非零转入清屏程序。
B4D7-  3AAF79    LD A,(79AF)    ；【汉字(及ASCII字符)显示】取汉字输出标志。
B4DA-  B7        OR A           ；测试。
B4DB-  C8        RET Z          ；无汉字输出则返回V2.0监控程序。
B4DC-  3AAF7A    LD A,(7AAF)    ；取输出缓冲区计数值。
B4DF-  B7        OR A           ；测试。
B4E0-  CAA7B6    JP Z,B6A7      ；缓冲区为空则返回时跳过监控程序的显示部分。
B4E3-  47        LD B,A         ；待显示字符数存入B作为循环变量。
B4E4-  21B27A    LD HL,7AB2     ；HL＝显示缓冲区首址。
B4E7-  2B        DEC HL         ；后退一字节。
B4E8-  C5        PUSH BC        ；存循环变量。
B4E9-  23        INC HL         ；指向缓冲区下一字节。
B4EA-  7E        LD A,(HL)      ；显示码取入A。
B4EB-  4F        LD C,A         ；存于C。
B4EC-  B7        OR A           ；测试显示码是否00H（正相汉字类型码）。
B4ED-  282E      JR Z,B51D      ；若是则转入正相汉字处理程序段。
B4EF-  FE01      CP 01          ；是不是01H（反相汉字类型码）？
B4F1-  2828      JR Z,B51B      ；若是则转入反相汉字处理程序段。
B4F3-  FE20      CP 20          ；是不是ASCII字符码？
B4F5-  3017      JR NC,B50E     ；是则转入ASCII字符处理程序段。
B4F7-  185C      JR B555        ；否则转去继续测试显示控制码。
B4F9-  00        NOP
B4FA-  00        NOP
B4FB-  00        NOP
B4FC-  00        NOP
B4FD-  2F        CPL            ；【汉字(及ASCII字符)显示后续段】A取反。
```

```
B4FE-  0601     LD B,01        ；B＝ASCII字符类型（即所占列数）码。
B500-  324978   LD (7849),A    ；(以上为ASCII符显示用)置显示相控制单元。
B503-  E5       PUSH HL        ；存缓冲区指针值。
B504-  CD23B5   CALL B523      ；显示该汉字或字符。
B507-  E1       POP HL         ；取回显示缓冲区指针值。
B508-  C1       POP BC         ；取回循环变量。
B509-  10DD     DJNZ B4E8      ；循环，输出显示其余显示码。
B50B-  C39DB6   JP B69D        ；显示完毕转入结束处理并回到V2.0监控程序。
B50E-  CB7F     BIT 7,A        ；【ASCII字符显示】测试是否反相字符。
B510-  3E00     LD A,00        ；A清零预置为正相。
B512-  28EA     JR Z,B4FE      ；若是正相显示码则转移（不改变A）。
B514-  CBB9     RES 7,C        ；将反相显示码的D7复位为0。
B516-  18E5     JR B4FD        ；转去将A变为FFH以使显示反相。
B518-  00       NOP
B519-  00       NOP
B51A-  00       NOP
B51B-  3EFF     LD A,FF        ；【汉字显示】(反相汉字入口)设为反相。
B51D-  23       INC HL         ；(正相汉字入口)指向缓冲区下一单元。
B51E-  4E       LD C,(HL)      ；将汉字显示码取入C。
B51F-  0602     LD B,02        ；设汉字类型（所占列数）码02H。
B521-  18DD     JR B500        ；输出显示。
B523-  78       LD A,B         ；【求字库地址】字符类型码取入A。
B524-  FE01     CP 01          ；是不是ASCII字符？
B526-  79       LD A,C         ；字符码存入A。
B527-  2600     LD H,00        ；H清零。
B529-  2017     JR NZ,B542     ；若是汉字则跳转。
B52B-  D620     SUB 20         ；ASCII码－20H，得到在ASCII字符表中的位序。
B52D-  6F       LD L,A         ；放入L，HL为字符在表中位序。
B52E-  E5       PUSH HL        ；存栈。
B52F-  0603     LD B,03        ；将HL值乘以8。
B531-  CB25     SLA L
B533-  CB14     RL H
B535-  10FA     DJNZ B531
B537-  D1       POP DE         ；取出原HL值。
```

```
B538-  19        ADD HL,DE      ; HL=位序数×9，即其在字库的相对首址。
B539-  11A05C    LD DE,5CA0     ; ASCII字库首址。
B53C-  19        ADD HL,DE      ; HL=字符点阵数据的首地址。
B53D-  3E01      LD A,01        ; 给以ASCII类型标志。
B53F-  182D      JR B56E        ; 转入行列位置处理。
B541-  00        NOP
B542-  6F        LD L,A         ; 【求汉字字库地址】字号放入L(H=0)。
B543-  0605      LD B,05        ; HL×32，求出在字库的相对首址。
B545-  CB25      SLA L
B547-  CB14      RL H
B549-  10FA      DJNZ B545
B54B-  ED5BE379  LD DE,(79E3)   ; 取用户字库首址。
B54F-  19        ADD HL,DE      ; HL=汉字点阵数据的首地址。
B550-  3E02      LD A,02        ; 给出汉字类型标志。
B552-  181A      JR B56E        ; 转入行列位置处理。
B554-  00        NOP
B555-  E5        PUSH HL        ; 【控制码输出显示】存缓冲区当前地址。
B556-  21B179    LD HL,79B1     ; HL=显示列指针单元地址。
B559-  FE0F      CP 0F          ; 测试当前字符是不是0FH(分段显示控制码)。
B55B-  2805      JR Z,B562      ; 若是则转去分段。
B55D-  FE0D      CP 0D          ; 是不是0DH(显示换行控制码)?
B55F-  20A6      JR NZ,B507     ; 是显示码之外的代码则跳过，去取下个代码。
B561-  37        SCF            ; Cy置位作为子程序中换行部分的分支条件。
B562-  CD3AB6    CALL B63A      ; 求出换行或分段后的下个起始列位置。
B565-  28A0      JR Z,B507      ; 若Z置位表示下一列正是段首，不作处理。
B567-  66        LD H,(HL)      ; 将当前列数放入H。
B568-  CDBAB5    CALL B5BA      ; 从当前列到下一个起始列间都输出空格。
B56B-  189A      JR B507        ; 继续输出缓冲区显示码。
B56D-  00        NOP
B56E-  E5        PUSH HL        ; 【换行及卷屏处理】存当前字库地址。
B56F-  F5        PUSH AF        ; 存字符类型。
B570-  ED5BB079  LD DE,(79B0)   ; 取行列指针值。
B574-  7B        LD A,E         ; 待显示行数放入E。
B575-  FE0C      CP 0C          ; 是否已超出屏幕下界（第11行）。
```

```
B577-  3802     JR C,B57B     ；未超出则跳过下一指令。
B579-  1E0B     LD E,0B       ；将显示下一字符的行改在屏幕末行。
B57B-  7A       LD A,D        ；取待显示列数。
B57C-  FE20     CP 20         ；列数是否已超出屏幕右边界（第31列）？
B57E-  3010     JR NC,B590    ；已超出则转移。
B580-  F1       POP AF        ；取回字符类型码（需占列数）。
B581-  F5       PUSH AF       ；再存。
B582-  82       ADD A,D       ；当前待显列数加待显字符所需列数。
B583-  FE20     CP 20         ；是否≥32?
B585-  384B     JR C,B5D2     ；若不到32则可转去显示汉字或ASCII字符。
B587-  2849     JR Z,B5D2     ；若为31+1=32则可转去显示ASCII字符。
B589-  D5       PUSH DE       ；为31+2=33时已不够显示汉字，存行列指针。
B58A-  0601     LD B,01       ；设计数器为1。
B58C-  CDC0B5   CALL B5C0     ；在当前显示位置补一ASCII空格。
B58F-  D1       POP DE        ；取回行列指针。
B590-  7B       LD A,E        ；行数取入A。
B591-  FE0B     CP 0B         ；测试是否是末行。
B593-  381C     JR C,B5B1     ；不是末行则换行而不卷动屏幕。
B595-  110040   LD DE,4000    ；末行换行时，以显示区首址为传送目标首址。
B598-  210042   LD HL,4200    ；以第二显示行的首址为传送的源首址。
B59B-  010016   LD BC,1600    ；传送范围为以下11行的全部点阵数据。
B59E-  00       NOP
B59F-  EDB0     LDIR          ；将第二行至末行的内容向上传送一行。
B5A1-  00       NOP
B5A2-  21B179   LD HL,79B1    ；HL=列指针单元地址。
B5A5-  AF       XOR A         ；A清零。
B5A6-  77       LD (HL),A     ；将列指针复位到0列。
B5A7-  0620     LD B,20       ；设计数器为32。
B5A9-  CDBCB5   CALL B5BC     ；输出32个空格将屏幕下端原有行变为空行。
B5AC-  57       LD D,A        ；D=列数(0)。
B5AD-  1E0B     LD E,0B       ；A=行数(11)。
B5AF-  1803     JR B5B4       ；跳转。
B5B1-  1C       INC E         ；(简单换行处理)行数增1。
B5B2-  1600     LD D,00       ；列数变0。
```

```
B5B4-  F1        POP AF          ；取字符类型码。
B5B5-  F5        PUSH AF         ；再存。
B5B6-  1819      JR B5D1         ；转入字符显示。
B5B8-  3E20      LD A,20         ；【行末清空】A=满行的列数(20)。
B5BA-  94        SUB H           ；减当前列数。
B5BB-  47        LD B,A          ；以应填充的空格数为循环变量。
B5BC-  ED5BB079  LD DE,(79B0)    ；取当前行列指针值。
B5C0-  C5        PUSH BC         ；存循环变量。
B5C1-  3E01      LD A,01         ；置为ASCII字符类型。
B5C3-  21E7AE    LD HL,AEE7      ；HL=ASCII空格码的点阵数据首址。
B5C6-  CD6EB5    CALL B56E       ；显示一空格。
B5C9-  C1        POP BC          ；取回循环变量。
B5CA-  10F0      DJNZ B5BC       ；循环显示将行的后部全部变为空格。
B5CC-  AF        XOR A           ；A清零。
B5CD-  C9        RET             ；返回。
B5CE-  00        NOP
B5CF-  E5        PUSH HL         ；【单字显示】存字库首址。
B5D0-  F5        PUSH AF         ；存字符类型码。
B5D1-  82        ADD A,D         ；A=显示当前字符后的下一个列数。
B5D2-  CD00B6    CALL B600       ；修改行列指针并计算当前字符的显示地址。
B5D5-  F1        POP AF          ；取回类型码。
B5D6-  D601      SUB 01          ；测试类型。
B5D8-  203E      JR NZ,B618      ；若是汉字则转移，是ASCII字符则A=0。
B5DA-  0606      LD B,06         ；【ASCII字符显示】设循环变量为6。
B5DC-  CD31B6    CALL B631       ；将A按屏幕基相和显示相位置为00H或FFH。
B5DF-  12        LD (DE),A       ；送入当前显示单元，显示一条空白线。
B5E0-  212000    LD HL,0020      ；将显示地址加32，对应于下一扫描行的同列。
B5E3-  19        ADD HL,DE
B5E4-  EB        EX DE,HL        ；地址放入DE。
B5E5-  10F8      DJNZ B5DF       ；循环6次显示出ASCII字符上部的空白部分。
B5E7-  E1        POP HL          ；取回字库首址。
B5E8-  0609      LD B,09         ；点阵部分共9字节。
B5EA-  7E        LD A,(HL)       ；取一字节点阵数据。
B5EB-  CD31B6    CALL B631       ；进行显示相位变换。
```

```
B5EE-  12        LD (DE),A      ；送入显示区目标单元。
B5EF-  23        INC HL         ；HL指向字库下一字节。
B5F0-  3E20      LD A,20        ；DE的显示地址加32指向对应下条扫描线地址。
B5F2-  83        ADD A,E
B5F3-  5F        LD E,A
B5F4-  3001      JR NC,B5F7
B5F6-  14        INC D
B5F7-  10F1      DJNZ B5EA      ；循环9次显示出字符的主体部分。
B5F9-  AF        XOR A          ；A清零。
B5FA-  CD31B6    CALL B631      ；对A进行显示相位转换。
B5FD-  12        LD (DE),A      ；在字符主体下面再显示一条空白线。
B5FE-  C9        RET            ；返回。
B5FF-  00        NOP
B600-  4A        LD C,D         ；【显示地址换算】当前列数放入C。
B601-  57        LD D,A         ；下一个列数放入D。
B602-  ED53B079  LD (79D0),DE   ；作为新的行列指针记存。
B606-  1600      LD D,00        ；D清零，DE＝行数。
B608-  0609      LD B,09        ；每行汉字占2⁹＝512个字节。
B60A-  CB23      SLA E          ；行数×512求出行首的相对地址。
B60C-  CB12      RL D
B60E-  10FA      DJNZ B60A      ；循环结束时B＝0。
B610-  EB        EX DE,HL       ；行首相对地址存于HL。
B611-  09        ADD HL,BC      ；加上列数为当前显示位置的相对地址，
B612-  010040    LD BC,4000     ；再加上显示区首址。
B615-  09        ADD HL,BC      ；HL＝显示地址。
B616-  EB        EX DE,HL       ；当前汉字在显示区的首地址放入DE。
B617-  C9        RET            ；返回。
B618-  E1        POP HL         ；【汉字单字显示】取字库地址。
B619-  0E10      LD C,10        ；C＝一个字的点阵行数。
B61B-  0602      LD B,02        ；B＝每行点阵的字节数。
B61D-  7E        LD A,(HL)      ；由字库取一字节点阵数据。
B61E-  CD31B6    CALL B631      ；进行相位变换。
B621-  12        LD (DE),A      ；送入左显示单元。
B622-  13        INC DE         ；指向右显示单元。
```

```
B623-  23       INC HL        ；指向字库下一字节。
B624-  10F7     DJNZ B61D     ；循环两次显示汉字中的一条线。
B626-  0D       DEC C         ；点阵行计数减1。
B627-  C8       RET Z         ；若已显示16行则返回。
B628-  3E1E     LD A,1E       ；A为上行右字节与下行左字节的距离。
B62A-  83       ADD A,E       ；显示地址增值指向下一点阵行左字节地址。
B62B-  5F       LD E,A
B62C-  30ED     JR NC,B61B
B62E-  14       INC D
B62F-  18EA     JR B61B       ；外循环。
B631-  E5       PUSH HL       ；【显示相位转换】存HL内容。
B632-  214978   LD HL,7849    ；HL＝显示相位标志单元地址。
B635-  AE       XOR (HL)      ；将当前显示点阵数据与相位标志值相异或。
B636-  23       INC HL        ；HL＝屏幕基相标志单元地址。
B637-  AE       XOR (HL)      ；再同显示基相码相异或，A即成所需显示相。
B638-  E1       POP HL        ；恢复HL内容。
B639-  C9       RET           ；返回。
B63A-  380D     JR C,B649     ；【终端列数设定】若是换行调用则转移。
B63C-  3E10     LD A,10       ；分段显示时令A＝后段首列数。
B63E-  BE       CP (HL)       ；与当前列数比较。
B63F-  D0       RET NC        ；在前段内则返回，若正应显示于此列则Z置位。
B640-  3E1F     LD A,1F       ；若已在后段，测试是否已到行末。
B642-  BE       CP (HL)
B643-  3002     JR NC,B647    ；若已应换行则令A＝32，使显示在下行行首。
B645-  3E2F     LD A,2F       ；在后段之内时令A＝48，使显示在下行后段。
B647-  3C       INC A         ；A加1，补足设定的终端列数并复位Z标志。
B648-  C9       RET           ；返回。
B649-  3E1F     LD A,1F       ；为换行将A置为31列。
B64B-  BE       CP (HL)       ；与当前列数比较。
B64C-  3002     JR NC,B650    ；若显示列数未满则终端列数为32。
B64E-  3E3F     LD A,3F       ；已到末列则终端列数为64，以清一个空行。
B650-  3C       INC A         ；补足设定的终端列数并使Z标志复位。
B651-  C9       RET           ；返回。
B652-  37       SCF           ；Cy置位。
```

```
B653-  C9        RET            ；返回。
B654-  CD022B    CALL 2B02      ；【RST工作程序】取目标行号。
B657-  E5        PUSH HL        ；存扫描指针值。
B658-  CD2C1B    CALL 1B2C      ；搜索目标行地址。
B65B-  0B        DEC BC         ；从行的首址后退一字节。
B65C-  ED43FF78  LD (78FF),BC   ；置为RAED语句读数指针。
B660-  E1        POP HL         ；恢复扫描指针值。
B661-  C9        RET            ；返回执行驱动程序。
B662-  CDAFB1    CALL B1AF      ；【高显清屏执行程序】4000H空间切换为VRAM。
B665-  110040    LD DE,4000     ；目标首址为4000H。
B668-  FE03      CP 03          ；测试代码是否为全屏清除。
B66A-  380B      JR C,B677      ；不是则转移。
B66C-  210000    LD HL,0000     ；将汉字行列指针复位。
B66F-  22B079    LD (79B0),HL
B672-  010018    LD BC,1800     ；共清除6KB。
B675-  180C      JR B683        ；转入操作。
B677-  FE02      CP 02          ；是否为清除中分辨率显示区（7000H～77FFH）？
B679-  3805      JR C,B680      ；不是则转移。
B67B-  010008    LD BC,0800     ；清除2KB。
B67E-  1803      LR B683        ；转入操作。
B680-  010002    LD BC,0200     ；只清除低分辨率显示区（7000H～71FFH）。
B683-  210300    LD HL,0003     ；正相清屏借用的显示数据(00H)源地址。
B686-  3A4A78    LD A,(78A4)    ；测试用户定义的屏幕基相。
B689-  B7        OR A
B68A-  2802      JR Z,B68E      ；若是正相则进行操作。
B68C-  2EF1      LD L,F1        ；否则借用00F1H的数据（FFH）为清屏码。
B68E-  EDA0      LDI            ；向目标地址送入一个清屏代码。
B690-  2B        DEC HL         ；仍指回清屏码原地址。
B691-  EA8EB6    JP PE,B68E     ；未送完规定字节则继续进行。
B694-  AF        XOR A          ；A清零。
B695-  324778    LD (7847),A    ；清除清屏标志。
B698-  324978    LD (7849),A    ；显示相标志置为正相。
B69B-  180A      JR B6A7        ；转去返回V2.0监控程序。
B69D-  21B27A    LD HL,7AB2     ；【显示结束】HL＝显示缓冲区首址。
```

```
B6A0-   22B07A    LD (7AB0),HL     ; 使显示缓冲区指针初始化。
B6A3-   AF        XOR A            ; A清零。
B6A4-   32AF7A    LD (7AAF),A      ; 清除显示缓冲区计数器。
B6A7-   C1        POP BC           ; 清除堆栈中PH扩充监控程序入口断点数据。
B6A8-   C3C52E    JP 2EC5          ; 跳过V2.0显示输出程序,进入扫描键盘阶段。
B6AB-   F7        RST 30           ; 【取十六进制数码值】取首字符。
B6AC-   3003      JR NC,B6B1       ; 非数字则跳转。
B6AE-   D630      SUB 30           ; 数字的ASCII码减30H得到本位值。
B6B0-   C9        RET              ; 返回。
B6B1-   FE41      CP 41            ; 代码是否≥41H。
B6B3-   389D      JR C,B652        ; ≯41H不是十六进制数码，以C状态返回。
B6B5-   FE47      CP 47            ; 是否<47H?
B6B7-   3099      JR NC,B652       ; 若≥47H不是十六进制数码，转取置位Cy返回。
B6B9-   D637      SUB 37           ; A~F的ASCII码各减37H得到本位值。
B6BB-   C9        RET              ; 返回。
B6BC-   F5        PUSH AF          ; 【RENUM工作程序】存后跟字符及测试状态。
B6BD-   E5        PUSH HL          ; 存扫描指针值。
B6BE-   211275    LD HL,7512       ; HL=RENUMBER程序接口标志地址(显示区内)。
B6C1-   CDA406    CALL 06A4        ; 检测是否装配有RENUMBER模块，有则转入。
B6C4-   C34A1E    JP 1E4A          ; 若未装RENUMBER模块为“非法调用”错误。
B6C7-   00        NOP
```

四、系统的组装和启动

由于扩充系统各模块地址不连续，所以作为几个文件分别装入内存目标区。为了简化操作，设置了一个系统装载程序。它的主体就是主程序的读带程序(因此时主程序尚不存在，无法利用其功能)，可自动将四个基本模块连锁调入，装到指定地址，并且装入一个系统初始化程序PH-1。磁带文件装载完毕后即转入初始化程序，启动系统工作。

作为一种加密措施，基本模块的磁带文件均以V2.0的D型文件记带，装载程序也以F1H作为目标文件类型标志。因为V2.0的CLOAD、CRUN和PH的各种磁带操作命令都不接受F2H标志的磁带文件，所以不易对它们进行解读。

(一)　系统装载程序(磁带文件名"PH1.3")

```
7AE9-  00                        NOP
7AEA-  00                        NOP
7AEB-  21F47A                    LD HL,7AF4        ; "DO NOT STOP THE TAPE"字符串地址。
7AEE-  CDA728                    CALL 28A7         ; 显示。
7AF1-  1823                      JR 7B16           ; 跳转。
7AF3-  00 44 4F 20 4E 4F 54 20                     ; 【数据表】
7AFB-  53 54 4F 50 20 54 48 45
7B03-  20 54 41 50 45 21 00
7B0A-  00 72 5D 78 52 79 7A 5C
7B12-  59 AB 00 00
7B16-  21D77B                    LD HL,7BD7        ; HL=装载暂停处理程序首址。
7B19-  22AD79                    LD (79AD),HL      ; 装入BASIC输入程序中的DOS出口。
7B1C-  3EC3                      LD A,C3           ; 设置JP指令操作码。
7B1E-  32AC79                    LD (79AC),A       ; 打开此出口，按BREAK键则可执行上述程序。
7B21-  2158AB                    LD HL,AB58        ; HL=扩充系统保留区首址-1。
7B24-  22B178                    LD (78B1),HL      ; 作为用户区软终端。
7B27-  113200                    LD DE,0032        ; DE=50。
7B2A-  CD831E                    CALL 1E83         ; 预置50字节字符串区。
7B2D-  3EFF                      LD A,FF           ; 令A为非零。
7B2F-  324C78                    LD (784C),A       ; 关闭读带提示信息显示功能。
7B32-  CDCD78                    CALL 7BCD         ; 将4000H空间切换为VRAM。
7B35-  210A7B                    LD HL,7B0A        ; HL=系统磁带文件装入目标地址表首址。
7B38-  E5                        PUSH HL           ; 存栈备用。
7B39-  CD8C35                    CALL 358C         ; 取文件名(因HL指向00H,为文件名缺省方式)。
```

```
7B3C-  01F220    LD BC,20F2      ；设定目标文件类型码F2H（V2.0的D型）。
7B3F-  CD607B    CALL 7B60       ；复制并修改V2.0读带程序，启动磁带。
7B42-  01FF05    LD BC,05FF      ；设文件计数器初值(5个文件)。
7B45-  E1        POP HL          ；取回文件装入目标地址表当前地址。
7B46-  5E        LD E,(HL)       ；送入DE。
7B47-  23        INC HL
7B48-  56        LD D,(HL)
7B49-  23        INC HL          ；指向下一个文件的装入首址。
7B4A-  E5        PUSH HL         ；存栈备用。
7B4B-  C5        PUSH BC         ；存计数器值。
7B4C-  ED53D779  LD (79D7),DE    ；置装载首址指针。
7B50-  CDE07B    CALL 7BE0       ；显示文件号后读入当前磁带文件。
7B53-  C1        POP BC          ；恢复计数器。
7B54-  10EF      DJNZ 7B54       ；循环装入下一文件，直至5个文件装载完毕。
7B56-  AF        XOR A           ；A清零。
7B57-  324C78    LD A,(784C)     ；恢复读带提示信息显示功能。
7B5A-  CDD77B    CALL 7BD7       ；停止磁带并将4000H空间切换为DOS。
7B5D-  C30072    JP 7200         ；转入系统初始化程序。
7B60-  C5        PUSH BC         ；【复制与修改V2.0读带程序】存类型码。
7B61-  216436    LD HL,3664      ；HL＝源程序块首址。
7B64-  11277A    LD DE,7A27      ；DE＝暂存区首址。
7B67-  015700    LD BC,0057      ；BC＝传送字节数。
7B6A-  EDB0      LDIR            ；送入暂存区。
7B6C-  EB        EX DE,HL        ；HL＝暂存区终址＋1。
7B6D-  36C9      LD (HL),C9      ；置入RET指令码。
7B6F-  23        INC HL          ；指向下一字节——将是出错处理程序首址。
7B70-  22487A    LD (7A48),HL    ；将此地址装入读带程序的有关指令中。
7B73-  224E7A    LD (7A4E),HL
7B76-  22737A    LD (7A73),HL
7B79-  227C7A    LD (7A7C),HL
7B7C-  EB        EX DE,HL        ；将此地址换入DE作为传送目标首址。
7B7D-  211137    LD HL,3711      ；HL＝V2.0装载出错处理程序首址。
7B80-  0E1D      LD C,1D         ；BC＝传送字节数。
```

```
7B82-   EDB0       LDIR              ；将出错处理程序复制到暂存区。
7B84-   21A67B     LD HL,7BA6        ；HL＝修改读带程序用源指令块地址。
7B87-   11577A     LD DE,7A57        ；DE＝修改目标地址。
7B8A-   0E08       LD C,08           ；BC＝传送字节数。
7B8C-   EDB0       LDIR              ；送入修改。
7B8E-   212A7A     LD HL,7A2A        ；HL＝再次进入的入口地址。
7B91-   228C7A     LD (7A8C),HL      ；修改有关指令。
7B94-   229A7A     LD (7A9A),HL
7B97-   C1         POP BC            ；取回目标文件类型码。
7B98-   ED43377A   LD (7A37),BC      ；送入读带程序中。
7B9C-   CDAE7B     CALL 7BAE         ；启动磁带。
7B9F-   C9         RET               ；返回。
7BA0-   F3         DI                ；【读带】关中断。
7BA1-   CD277A     CALL 7A27         ；调用修改后的读带程序装入磁带文件。
7BA4-   FB         EI                ；开中断。
7BA5-   C9         RET               ；返回。
7BA6-   ED5BD779   LD DE,(79D7)      ；【修改用的源指令块】将指定的首址装入DE。
7BAA-   00         NOP
7BAB-   00         NOP
7BAC-   00         NOP
7BAD-   00         NOP
7BAE-   21F621     LD HL,21F6        ；【系统软开关操作】参见PH-5的B190H～B1AEH。
7BB1-   1809       JR 7BBC
7BB3-   21E6FE     LD HL,FEE6
7BB6-   CDBC7B     CALL 7BBC
7BB9-   21F620     KD HL,20F6
7BBC-   F5         PUSH AF
7BBD-   3A3B78     LD A,(783B)
7BC0-   22C37B     LD (7BC3),HL
7BC3-   E6FD       AND FD
7BC5-   323B78     LD (783B),A
7BC8-   320068     LD (6800),A
7BCB-   F1         POP AF
```

```
7BCC-  C9          RET
7BCD-  21F602      LD HL,02F6
7BD0-  18EA        JR 7BBC
7BD2-  21E6FD      LD HL,FDE6
7BD5-  18E5        JR 7BBC
7BD7-  AF          XOR A           ;【装载暂停处理程序】A清零。
7BD8-  324C78      LD (784C),A     ；恢复读带提示信息功能。
7BDB-  CDB37B      CALL 7BB3       ；停止磁带。
7BDE-  18F2        JR 7BD2         ；转去将4000H空间切换到DOS后返回。
7BE0-  3E36        LD A,36         ；【显示文件编号】令A＝36。
7BE2-  90          SUB B           ；减B中的文件计数值，得到1～5反白字符码。
7BE3-  32DE71      LD (71DE),A     ；将字符码显示在屏幕倒数第二行末。
7BE6-  C3A07B      JP 7BA0         ；转入读取磁带文件。
```

（二）系统初始化程序（磁带文件名“PH-1”）

```
7200-  214E78      LD HL,784E      ；HL＝系统工作区的目标段首址。
7203-  060F        LD B,0F         ；范围15字节。
7205-  77          LD (HL),A       ；进入本程序时A＝0，清除一字节。
7206-  23          INC HL          ；指向下一字节。
7207-  10FC        DJNZ 7205       ；循环清除。
7209-  2A0478      LD HL,(7804)    ；HL＝RST 10H子程序当前入口地址。
720C-  112BAD      LD DE,AD2B      ；DE＝PH系统的RST 10H入口地址。
720F-  DF          RST 18          ；二者比较，作为系统识别。
7210-  2807        JR Z,7219       ；若相同则原在PH系统中，不作处理。
7212-  222B78      LD (782B),HL    ；否则将原系统的入口存贮在系统工作区。
7215-  ED530478    LD (7804),DE    ；置换为本系统的入口指针。
7219-  AF          XOR A           ；A清零。
721A-  322A78      LD (782A),A     ；置图形显示相为正相。
721D-  32AF79      LD (79AF),A     ；清除汉字显示标志。
7220-  324778      LD (7847),A     ；清除清屏标志。
7223-  32E279      LD (79E2),A     ；汉字库字数预置为1个（字号0）。
7226-  211DB3      LD HL,B31D      ；HL＝清除汉字显示标志子程序入口地址。
7229-  22A779      LD (79A7),HL    ；装入错误处理程序的DOS出口以增加此内容。
722C-  21A679      LD HL,79A6      ；指向这个DOS出口的首字节。
722F-  36C3        LD (HL),C3      ；置入JP指令操作码将出口打开。
```

```
7231-  210000     LD HL,0000      ; HL＝0行0列。
7234-  22B079     LD (79B0),HL    ; 作为汉字显示位置指针值。
7237-  2159AB     LD HL,AB59      ; HL＝“汉”字点阵数据首址。
723A-  22E379     LD (79E3),HL    ; 作为汉字库首址指针值。
723D-  2E78       LD L,78         ; HL＝AB78H。
723F-  22AA79     LD (79AA),HL    ; 作为汉字库终端指针值。
7242-  21A7B2     LD HL,B2A7      ; HL＝停止磁带子程序入口地址。
7245-  22AD79     LD (79AD),HL    ; 装入BASIC输入程序中的DOS出口。
7248-  21AC79     LD HL,79AC      ; 指向这个出口的首字节。
724B-  36C3       LD (HL),C3      ; 将出口打开，回到输入阶段时磁带自停。
724D-  216DAE     LD HL,AE6D      ; HL＝PH扩充监控程序入口地址。
7250-  227E78     LD (787E),HL    ; 作为V2.0中断服务程序的用户出口向量。
7253-  21781D     LD HL,1D78      ; HL＝V2.0的“取下一字符”子程序入口。
7256-  221078     LD (7810),HL    ; 作为RST 30H的入口向量，以代行其功能。
7259-  210F78     LD HL,780F      ; 指向这个向量的首字节。
725C-  36C3       LD (HL),C3      ; 装入JP指令操作码。
725E-  217D78     LD HL,787D      ; HL＝中断服务程序用户出口向量首字节。
7261-  36C3       LD (HL),C3      ; 将出口打开准备运行扩充的监控程序。
7263-  FB         EI              ; 开中断。
7264-  CD4A1B     CALL 1B4A       ; 将BASIC指针初始化。
7267-  217072     LD HL,7270      ; HL＝版型标识字符串首址。
726A-  CDA728     CALL 28A7       ; 显示版型标识。
726D-  C3191A     JP 1A19         ; 转入BASIC输入程序，等待输入。
7270-  4C 41 53 45 52 20 32 30    ; 【数据表】
7278-  30 2D 33 31 30 0D 54 4F
7280-  53 20 42 41 53 49 43 20
7288-  50 48 31 2E 33 0D 0D 00
```

（三） 扩充模块

系统配置的扩充模块有重编行号程序（“RENUMBER”），造型表功能程序（“DRAW”），字库建立和编辑程序（“CREATA/EDIT”，简称“C/E”）等。下面以“C/E”程序为例进行解说。

“C/E”程序是用BASIC和汇编语言结合编制的。采取这种形式，主要是为了提供一个应用PH系统的示例。BASIC 是主程序，除作为总框架外，主要是用其所长处理显示提示、扫描键盘和数学运算等。而汉字点阵数据的处理工作则由机器语言子程序负担。BASIC 程序除调用机器语言子程序外，还负责向它们送入必需的参数。二者互补配合的结果，程序规模不大（约1.5KB），功能相当可观，速度也令人满意。

由于机器语言子程序都存放在后移变量区所腾出的保留区中，所以用户对 BASIC 文本不能随意增删，以免改变了子程序的入口地址。BASIC 部分请读者自行阅读，只解说汇编部分。

C/E 程序的 BASIC 文本中字符间实际均未留空格。

BASIC 部分程序

```
5 INPUT "NO.MOST(<=255)";H$ : HGR : IF H$="" THEN 30 ELSE H=VAL(H$)
10 IF H>255 OR H<0 ERROR 6
20 POKE -32561,H : CALL 32974 : CLEAR 50 : CALL 32832
30 H=PEEK(31202) : GOSUB 300
40 A=PEEK(31203)+PEEK(31204)*256 : K=30873
45 POKE 30796,0 : X=0 : Y=0 : GOSUB 170
47 HPRINT@(11,24); " "
50 B=0 : C=0 : N=0
55 POKE K,0 : C=PEEK(K) : IF C=0 THEN 55 ELSE IF C>47 AND C<58 THEN 215
60 IF C=32 IF Y<11 Y=Y+1 : GOTO 150 ELSE Y=0 : GOTO 95
70 IF C=77 IF X>0 X=X-1 : GOTO 150 ELSE X=15 : GOTO 120
80 IF C=44 IF X<15 X=X+1 : GOTO 150 ELSE X=0 : GOTO 130
90 IF C=46 IF Y>0 Y=Y-1 : GOTO 150 ELSE Y=11 : GOTO 140 ELSE 200
95 IF X<15 X=X+1 : GOTO 150 ELSE X=0 : GOTO 150
120 IF Y>0 Y=Y-1 : GOTO 150 ELSE Y=11 : GOTO 150
130 IF Y<11 Y=Y+1 : GOTO 150 ELSE Y=0 : GOTO 150
140 IF X>0 X=X-1 ELSE X=15
150 GOSUB 160 : GOTO 55
160 CALL 32766
170 POKE 30807,X : POKE 30806,Y : CALL 32766 : RETURN
```




```
200 IF C=58 THEN 220 ELSE IF C=45 HGR:GOTO 30
202 IF C=13 HGR:HPRINT@(11,25);"LOADD":CALL 32736:E=0:GOTO 45
205 IF C=80 GOSUB 170:HPRINT@(11,25);"LOADW":LOADW# ELSE 55
210 H=PEEK(31202):HPRINT@(11,0);" ":GOTO 40
215 C=C-48:B=B×10+C:N=N+1:HPRINT@(11,24);B:IF N<4 THEN 55 ELSE 230
220 IF N=0 B=D:HPRINT@(11,24);B
230 IF B>H THEN HPRINT@(11,25);"?":GOTO 50
232 HPRINT@(11,30);[B]; :HPRINT@(11,30);#[B]; :POKE -32661,B:D=B
235 POKE K,0:C=PEEK(K):IF C=0 THEN 235 ELSE IF C=58 THEN 250
237 IF C=76 CALL 32892:GOTO 240 ELSE IF C=59 CALL 32914:GOTO 232
239 IF C=62 THEN 240 ELSE IF C=42 N=0:D=D-(D=0)-1:GOTO 220 ELSE 47
240 B=B×32+A:POKE 30809,INT(B/256):POKE 30808,B-INT(B/256)×256
245 CALL 32766:CALL 32790:CALL 32766:HPRINT@(11,30);[D];
250 D=D+1:IF C=62 THEN 50 ELSE N=0:GOTO 220
300 IF H<176 HPRINT[0\H]; :HPRINT@(11,2);"0 -";H:RETURN
310 IF E=0 HPRINT[0\127]:HPRINT@(11,2);"0 -127":E=1:RETURN
320 E=0:HPRINT[128\H]:HPRINT@(11,1);128;" -";H:RETURN
```

汇编部分程序

```
7FE0-  00       NOP            ;【读取基本字库文件】
7FE1-  21E07F   LD HL,7FE0     ;扫描指针指向零字节。
7FE4-  CD8C35   CALL 358C      ;取文件名（文件名缺省方式）。
7FE7-  3EFF     LD A,FF        ;A为不显示提示信息控制码。
7FE9-  324C78   LD (784C),A    ;关闭读带提示信息显示功能。
7FEC-  210040   LD HL,4000     ;以4000H为装入首址。
7FEF-  22D779   LD (79D7),HL   ;作为装载首址指针。
7FF2-  3E2B     LD A,2B        ;设置为指定装载地址方式。
7FF4-  3233B0   LD (B033),A    ;置入读带程序。
7FF7-  01F220   LD BC,20F2     ;指定目标文件类型为V2.0的D型。
7FFA-  C30DB0   JP B00D        ;转入读带程序。
7FFD-  00       NOP            ;【光标所在汉字反相显示】
7FFE-  CD3480   CALL 8034      ;计算当前光标所在汉字的首地址。
8001-  EB       EX DE,HL       ;将字的首址存入HL。
8002-  0610     LD B,10        ;设行计数器初值（每字点阵分16行）。
```

```
8004-  111F00   LD DE,001F     ；DE＝行间地址增量。
8007-  76       HALT           ；暂停，等待FS信号有效后继续。
8008-  7E       LD A,(HL)      ；取当前行左字节。
8009-  EE7F     XOR 7F         ；将D6～D0各位取反。
800B-  77       LD (HL),A      ；送回原单元。
800C-  23       INC HL         ；指向本行右字节。
800D-  7E       LD A,(HL)      ；取入A。
800E-  EEFE     XOR FE         ；将D7～D1各位取反。
8010-  77       LD (HL),A      ；送回原单元。
8011-  19       ADD HL,DE      ；求出下一行显示首址。
8012-  10F4     DJNZ 8008      ；循环16次，处理32字节。
8014-  C9       RET            ；返回。
8015-  00       NOP            ；【将显示汉字数据送入用户字库区】
8016-  CD3480   CALL 8034      ；计算光标所指汉字的显示区地址。
8019-  2A5878   LD HL,(7858)   ；取用户字库指针。
801C-  EB       EX DE,HL       ；DE＝字库指针，HL＝显示地址。
801D-  0E10     LD C,10        ；设外循环计数器（16行）。
801F-  0602     LD B,02        ；设内循环计数器（2字节）。
8021-  7E       LD A,(HL)      ；由显示区取一字节。
8022-  12       LD (DE),A      ；送入字库。
8023-  13       INC DE         ；指向字库下一单元。
8024-  23       INC HL         ；指向显示区下一字节。
8025-  10FA     DJNZ 8021      ；循环，传送下一字节。
8027-  3E1E     LD A,1E        ；A＝行间增量。
8029-  85       ADD A,L        ；将HL指向下一行左字节。
802A-  6F       LD L,A
802B-  3E00     LD A,00
802D-  8C       ADC A,H
802E-  67       LD H,A
802F-  0D       DEC C          ；外循环计数器减1。
8030-  20ED     JR NZ,801F     ；循环16次，传送32字节。
8032-  C9       RET            ；返回。
8033-  00       NOP            ；【计算显示区地址】
```

```
8034-  ED5B5678   LD DE,(7856)     ; 取光标位置（X、Y值）。
8038-  7A         LD A,D           ; 令A=X。
8039-  87         ADD A,A          ; X×2=当前的列数。
803A-  4F         LD C,A           ; 列数存于C。
803B-  C306B6     JP B606          ; 转入显示地址换算子程序。
803E-  00         NOP              ; 【将用户字库空区变为空格稿纸式样】
803F-  00         NOP
8040-  CDC280     CALL 80C2        ; 求出字库中末字的地址。
8043-  ED5BE379   LD DE,(79E3)     ; 取字库指针值。
8047-  DF         RST 18           ; 二者比较。
8048-  CCAC80     CALL Z,80AC      ; 若仅有一个字(例如初始状态)则先行处理。
804B-  210000     LD HL,0000       ; HL=新定义字库首址(由置指针子程序填入)。
804E-  22E379     LD (79E3),HL     ; 置为字库指针。
8051-  DF         RST 18           ; 与原字库比较。
8052-  D0         RET NC           ; 若新库小于原库则返回。
8053-  E5         PUSH HL          ; 存新字库首址。
8054-  B7         OR A             ; 清除Cy。
8055-  EB         EX DE,HL         ; HL=原库首址，DE=新库首址。
8056-  ED52       SBC HL,DE        ; 求增加的字节数。
8058-  0605       LD B,05          ; 字节数除32，得到字数。
805A-  CB3C       SRL H
805C-  CB1D       RR L
805E-  10FA       DJNZ 805A
8060-  4D         LD C,L           ; 增加的字数放入计数器C。
8061-  E1         POP HL           ; 取回字库首址。
8062-  CDAC80     CALL 80AC        ; 将新增的字库区变成空格稿纸式样。
8065-  0D         DEC C            ; 计数减1。
8066-  20FA       JR NZ,8062       ; 循环处理全部空区。
8068-  C9         RET              ; 返回。
8069-  00         NOP              ; 【计算字库地址】
806A-  1E00       LD E,00          ; 当前字号（主程序填在806BH）装入E。
806C-  1600       LD D,00          ; DE=字号。
806E-  0605       LD B,05          ; 字号×32。
```

```
8070-  CB23        SLA E
8072-  CB12        RL D
8074-  10FA        DJNZ 8070          ；循环结束时DE＝字库相对地址。
8076-  2AE379      LD HL,(79E3)       ；取字库指针值。
8079-  19          ADD HL,DE          ；求出绝对地址。
807A-  EB          EX DE,HL           ；DE＝该字点阵数据首地址。
807B-  C9          RET                ；返回。
807C-  CD6A80      CALL 806A          ；【在用户字库中插入字】求字库当前地址。
807F-  CDC280      CALL 80C2          ；求末字地址。
8082-  DF          RST 18             ；比较。
8083-  C8          RET Z              ；若在末字位置则返回。
8084-  E5          PUSH HL            ；存末字地址。
8085-  ED52        SBC HL,DE          ；求出高端字节数。
8087-  44          LD B,H             ；字节数放入BC，作为传送计数器。
8088-  4D          LD C,L
8089-  E1          POP HL             ；取回末字首址。
808A-  2B          DEC HL             ；退到倒数第二个字的末址，作为传送首址。
808B-  ED5BAA79    LD DE,(79AA)       ；以字库终址作为减址传送的目标首址。
808F-  EDB8        LDDR               ；将待插入处以下的汉字数据后移32字节，以腾
8091-  C9          RET                  出放置插入字的位置(末字被覆盖)。返回。
8092-  CD6A80      CALL 806A          ；【在用户字库中删除字】求待删字的首地址。
8095-  212000      LD HL,0020         ；HL＝一个汉字数据字节数。
8098-  19          ADD HL,DE          ；HL＝下一个字的首址。
8099-  E5          PUSH HL            ；存下字首址。
809A-  D5          PUSH DE            ；存待删字首址。
809B-  EB          EX DE,HL           ；下字首址放入DE。
809C-  2AAA79      LD HL,(79AA)       ；取字库终址。
809F-  23          INC HL             ；指向终址后一单元。
80A0-  B7          OR A               ；清除Cy。
80A1-  ED52        SBC HL,DE          ；求出高端字节数。
80A3-  44          LD B,H             ；字节数放入BC，作为传送计数器。
80A4-  4D          LD C,L
80A5-  D1          POP DE             ；取回待删字首址作为目标首址。
```

```
80A6-  E1       POP HL          ；取回下字首址作为源首址。
80A7-  2802     JR Z,80AB       ；若仅有一字则不必传送。
80A9-  EDB0     LDIR            ；将下字开始的全部数据向前传送32字节。
80AB-  EB       EX DE,HL        ；HL＝原来末字首址。
80AC-  3EFF     LD A,FF         ；【变空格】A为横线的点阵数据。
80AE-  77       LD (HL),A       ；置入首行左字节。
80AF-  23       INC HL          ；指向第二单元。
80B0-  77       LD (HL),A       ；置入首行右字节。
80B1-  23       INC HL          ；指向下一单元。
80B2-  060E     LD B,0E         ；中间14行的处理。
80B4-  3680     LD (HL),80      ；左字节为80H（左边显示一个点）。
80B6-  23       INC HL          ；指向下一单元。
80B7-  3601     LD (HL),01      ；右字节为01H（右边显示一个点）。
80B9-  23       INC HL          ；指向下一单元。
80BA-  10F8     DJNZ 80B4       ；循环14次。
80BC-  77       LD (HL),A       ；将末行也变为一条横线。
80BD-  23       INC HL
80BE-  77       LD (HL),A
80BF-  23       INC HL
80C0-  C9       RET             ；返回。
80C1-  00       NOP             ；【求末字首址】
80C2-  2AAA79   LD HL,(79AA)    ；取字库终址。
80C5-  011F00   LD BC,001F      ；BC＝31。
80C8-  B7       OR A            ；清除Cy。
80C9-  ED42     SBC HL,BC       ；HL＝末字首址。
80CB-  C9       RET             ；返回。
80CC-  00       NOP             ；【置指针】
80CD-  00       NOP
80CE-  110000   LD DE,0000      ；将用户定义的最大字号（80CFH中）取入E。
80D1-  7B       LD A,E          ；A＝最大字号。
80D2-  32E279   LD (79E2),A     ；记入字库字数指针。
80D5-  0605     LD B,05         ；字号×32。
80D7-  CB32     SLA E
```

```
80D9-  CB12     RL D
80DB-  10FA     DJNZ 80D7       ；循环结束时DE＝字库所需字节数－32。
80DD-  CDC280   CALL 80C2       ；求末字地址。
80E0-  2B       DEC HL          ；后退到倒数第二字的末址。
80E1-  B7       OR A            ；清除Cy。
80E2-  ED52     SBC HL,DE       ；HL＝字库区低端之前的地址。
80E4-  22B178   LD (78B1),HL    ；作为用户区软终端。
80E7-  23       INC HL          ；指向字库首址。
80E8-  224C80   LD (804C),HL    ；将字库首址填入清库子程序。
80EB-  C9       RET             ；返回。
```

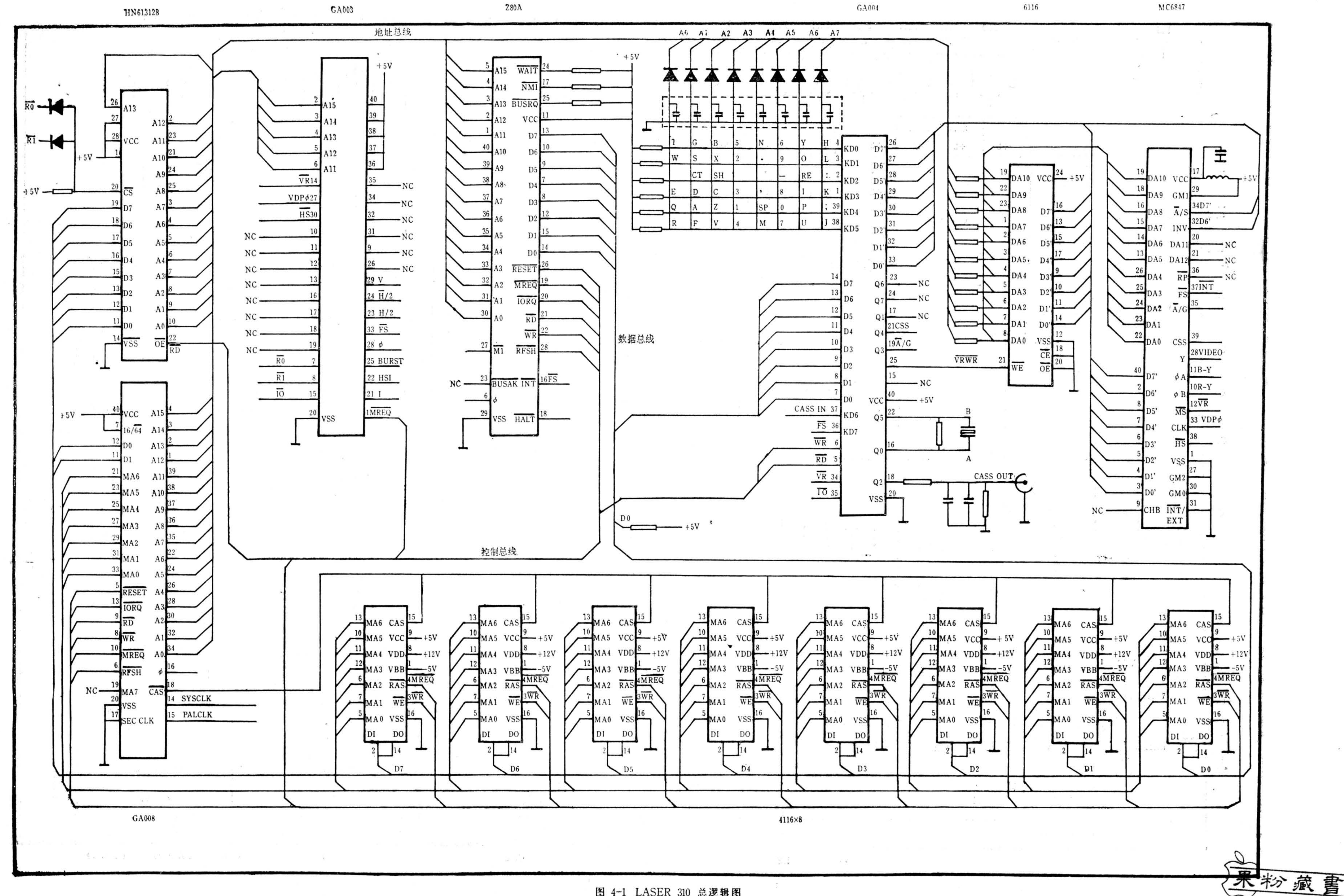


图 4-1 LASER 310 总逻辑图

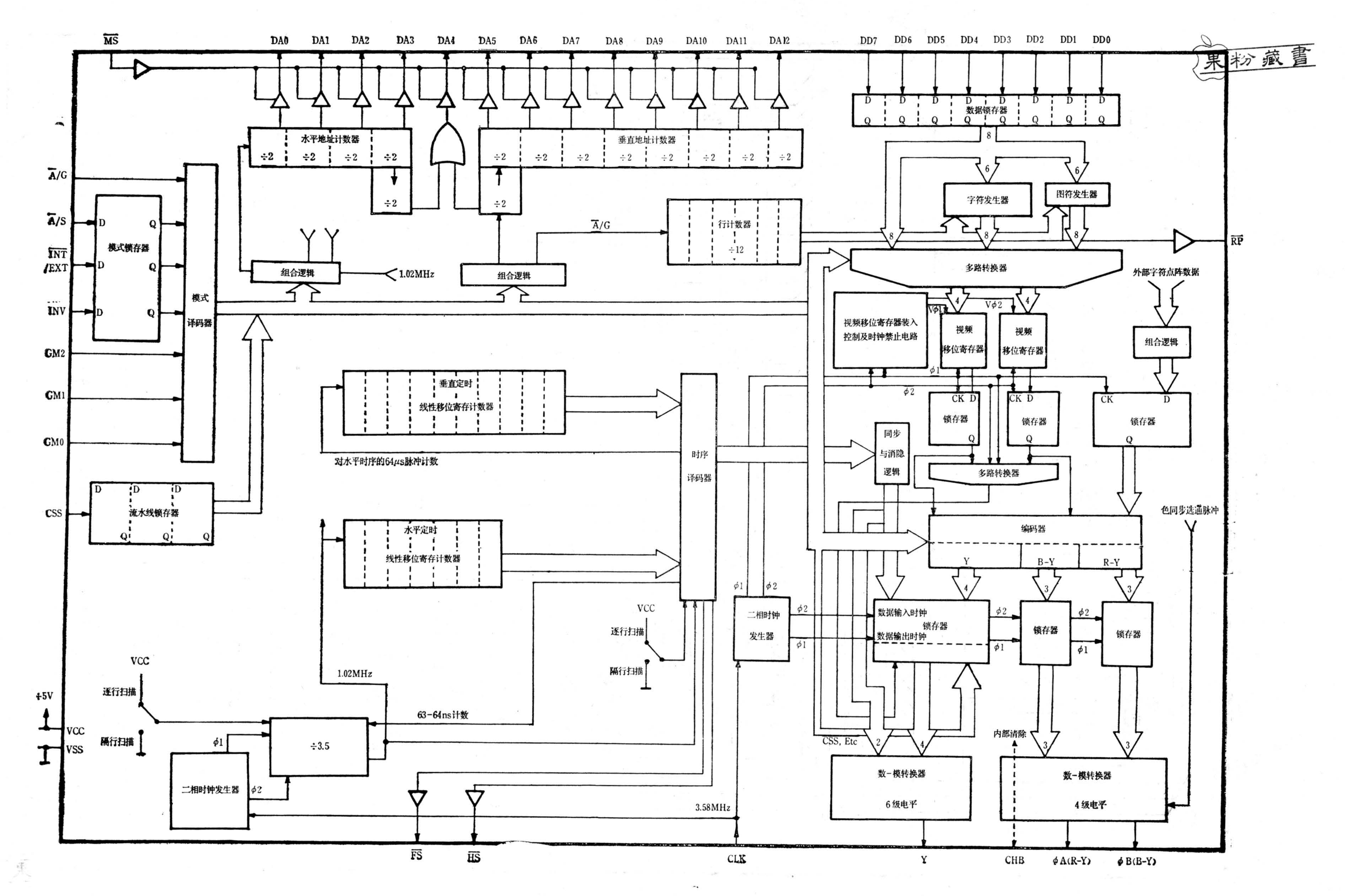


图 4-19 MC6847逻辑图

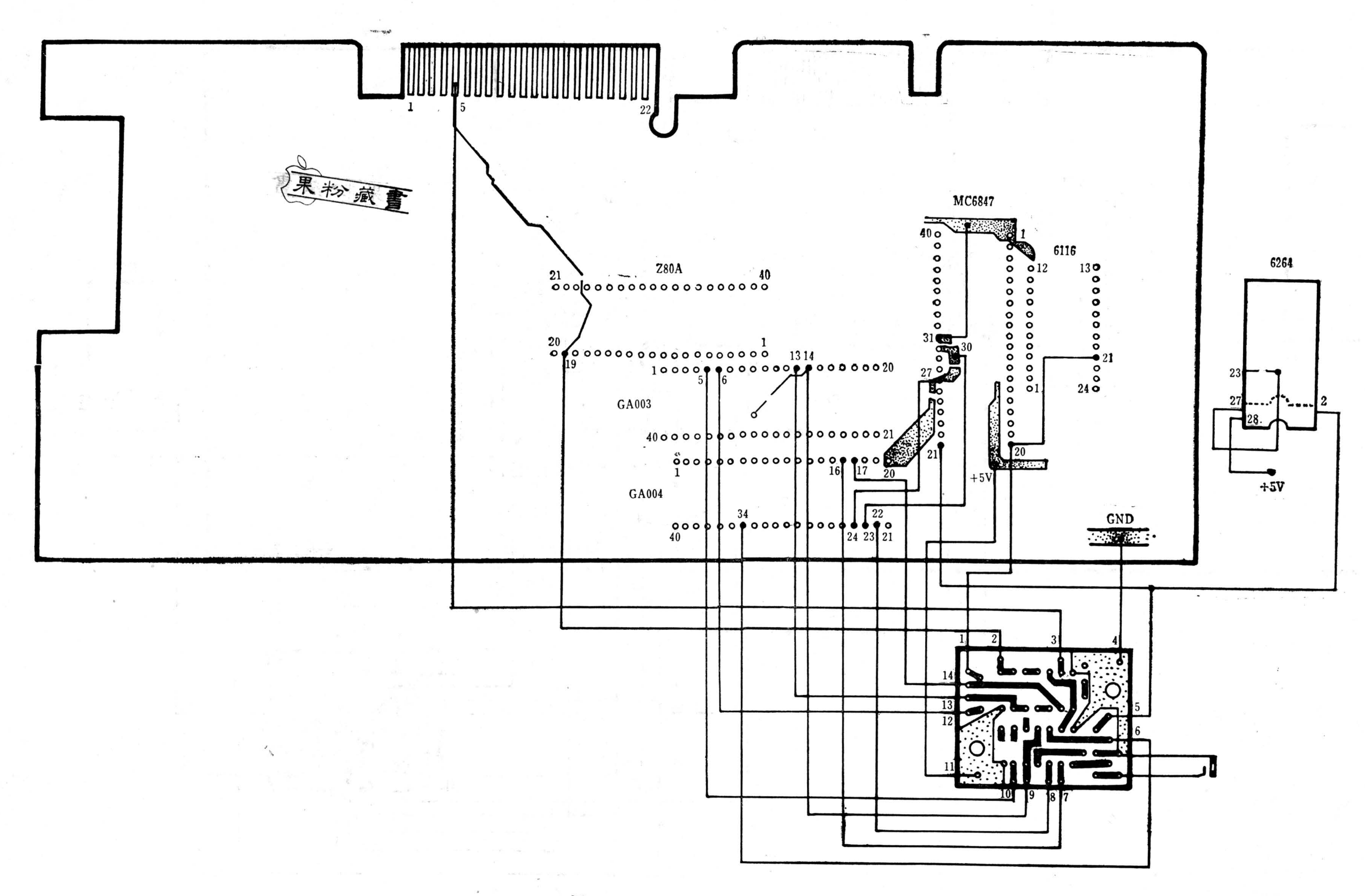


图 8-5 LASER 310电路改装图

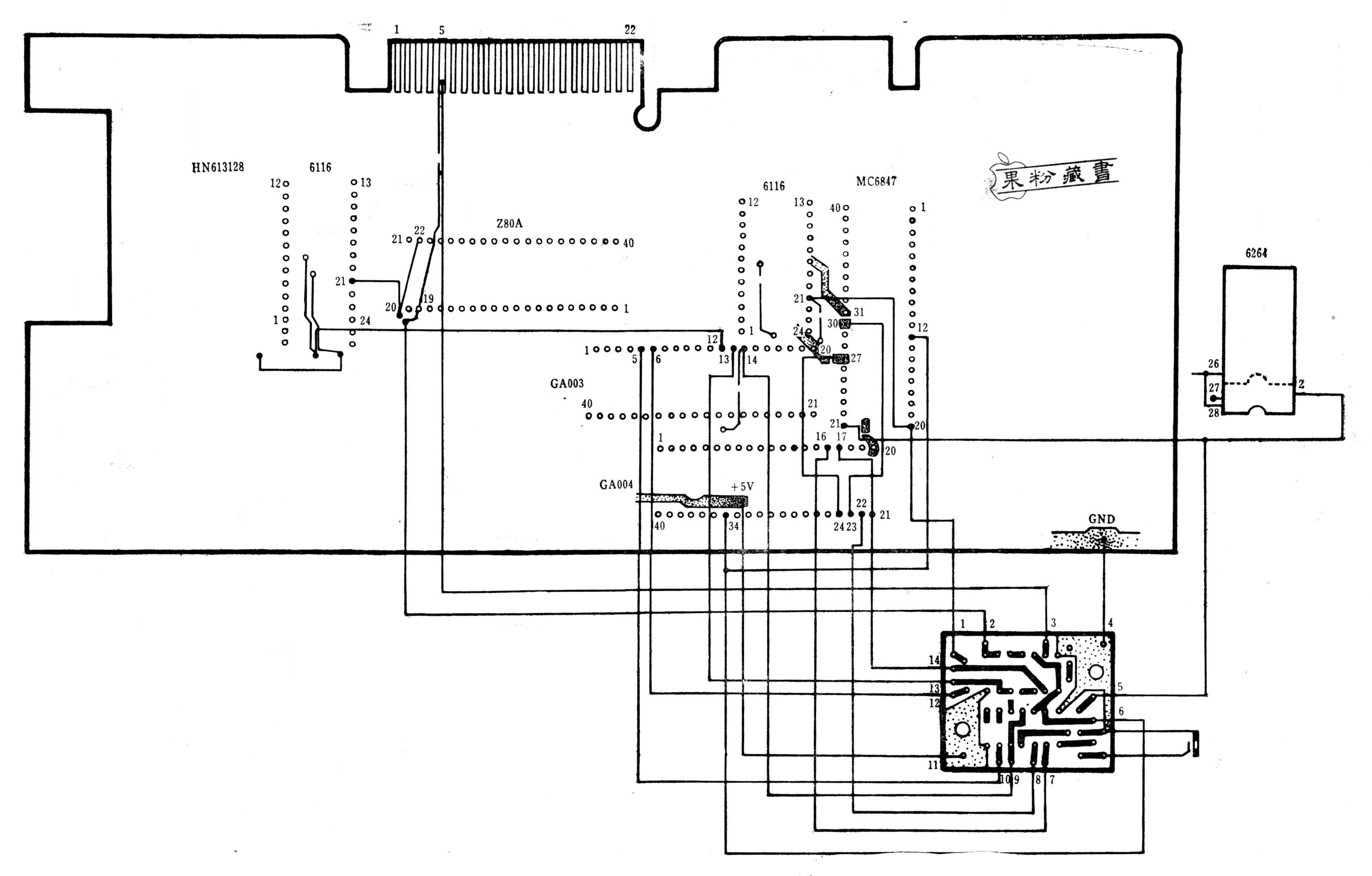


图 8-7 LASER 200 电路改装图

后　记

在中国青年出版社领导和同志们的努力下，本书得以呈现在读者面前，了却作者几年来的一桩心愿——同电脑初学者透彻地"侃"一"侃"计算机。

对本书的写作，贵州省科协、省青少年科技活动中心和成都《软件报》编辑部给予了十分可贵的支持和帮助。写作中参阅了国内外多种计算机科技书刊以及泽洛格、莫托洛拉、伟易达等厂家的技术资料，并有所引用。鉴于本书性质体例，不便一一列举，谨在此一并向有关作者和出版者深致谢意。为本书提供宝贵资料，编写调试程序，整理反汇编文本以及审校部分章节的有曾祥光、李革、谢雨平、邓石、彭海棣等同志。书中插图由杨毅、龚菊华、王利军、刘茗茗等绘制和摄影。

读者若需要书中介绍的LASER　310扩充系统软件(如PH系列、一、二级汉字库、DEBUG等）的磁带或软磁盘，可向贵州省科协青少年科技活动中心联系购买。地址：贵阳市青云路183号。邮政编码：550002。

为了让具有中等以上文化水平的读者都能读懂本书，写作时比较注重普及性和实用性，但在普及性和严谨性、实用性和系统性之间如何兼顾的问题上，解决得尚未能尽如人意，由于作者学识所限，书中错漏之处在所难免，敬希读者不吝赐教是幸。

作者

1992年2月



（京）新登字083号

责任编辑：黄大卫
封面设计：沈云瑞



微电脑剖析及功能改进
彭辛岷 著
*
中国青年出版社出版 发行
社址：北京东四12条21号 邮政编码：100708
中国青年出版社印刷厂印刷 新华书店经销
*
787×1092 1/16 18.75印张 2插页 440千字
1993年1月北京第1版 1993年1月北京第1次印刷
印数1—3,000册 定价9.60元
ISBN 7-5006-1169-2/TP·3

ISBN 7—5006—1169—2
TP·3　　定价9.60元